Introduction to App Development

Build and Publish Apps on any Platform

Daniel Walasinski

For High School Computer Science Teachers & Learners

Copyright © 2026 by Daniel Edward Walasinski

All rights reserved.

No portion of this book may be reproduced in any form without written permission from the publisher or author, except as permitted by U.S. copyright law.

Computer Science Standards Alignment

AP CSP Enduring Understandings - Evidenced

EU code	Status	Where it's covered
CRD-1	Evidenced	Lessons 1, 2, 8, 9, and 10 ask students to plan app ideas, choose layouts, make UI decisions, and create user-facing artifacts.
CRD-2	Evidenced	Lessons 1, 2, 4, 9, 12, 13, and 14 use a build, test, revise, device-test, and publish workflow.
AAP-1	Evidenced	Lessons 1, 2, 4, 5, 6, 7, 9, and 11 introduce variables/state, expressions, functions, event handlers, and app logic.
AAP-2	Evidenced	Lessons 2, 5, 6, 7, and 11 use conditionals, loops/intervals, input handling, evaluation logic, and game-state updates.
AAP-3	Evidenced	Lessons 2, 3, 7, 9, 10, 11, and 12 use components, procedures, screens, files, assets, and reusable code structures.
AAP-4	Evidenced	Lessons 2, 3, 4, 9, and 12 use Expo, React Native, SDKs, packages, APIs, storage, and framework-based development.

AP CSP - Not Evidenced

CSN-1, CSN-2, DAT-1, DAT-2, IOC-1, IOC-2

AP CSP Supplements

Standard(s)	Suggested supplement
CSN-1, CSN-2	Add a mini-unit on internet structure, protocols, DNS, routing, cybersecurity, encryption, and network reliability.
DAT-1, DAT-2	Add a data project using survey results, app logs, or API responses to represent, clean, visualize, compress, and interpret data.
IOC-1, IOC-2	Add reflection lessons on privacy, bias, accessibility, intellectual property, legal/policy issues, and social impacts of apps.

CSTA Level 2 - Evidenced

CSTA ID	Status	Where it's covered
2-CS-01	Partial	Lessons 8, 9, 10, and 11 ask students to analyze layout, spacing, safe areas, screen design, and interaction design.
2-CS-03	Evidenced	Lessons 1, 2, 4, 5, 6, 12, 13, and 14 repeatedly require students to identify and fix setup, terminal, preview, device-testing, and build problems.
2-AP-11	Evidenced	Lessons 1, 2, 5, 6, 7, and 11 use variables/state for display values, elapsed time, laps, player location, levels, and settings.
2-AP-12	Evidenced	Lessons 2, 5, 6, 7, and 11 combine conditionals, loops/intervals, event handlers, functions, and game-state updates.
2-AP-13	Evidenced	Lessons 3, 7, 9, 10, 11, and 12 break apps into screens, components, files, UI areas, functions, assets, and publishing steps.
2-AP-14	Evidenced	Lessons 2, 5, 6, 7, and 11 include reusable functions/components in React Native, JavaScript, Python, Swift, and UI-code workflows.
2-AP-15	Evidenced	Lessons 1, 2, 8, 9, and 12 ask students to test apps, compare screenshots to goals, request revisions, and improve behavior/usability.
2-AP-16	Partial	Lessons 2, 3, 4, 9, and 12 incorporate existing code, SDKs, Expo libraries, APIs, generated code, and assets; attribution instruction should be added.
2-AP-17	Evidenced	Lessons 1, 2, 4, 11, 12, 13, and 14 require running apps, testing buttons/screens, fixing errors, smoke testing devices, and checking builds.
2-AP-19	Evidenced	Lessons 2, 3, 12, and 13 model documentation through comments, screenshots, README/API notes, release notes, metadata, and debug records.
2-IC-20	Partial	Lessons 1, 5, 7, 10, 12, 13, and 14 compare development platforms and publishing pathways; add explicit life/career tradeoff reflection.
2-IC-21	Partial	Lessons 3, 8, 9, and 10 address accessibility through touch targets, spacing, readable text, and responsive design; expand accessibility discussion.
2-IC-23	Partial	Lessons 12, 13, and 14 include permissions, privacy policies, app metadata, and store requirements; add explicit public/private information discussion.

CSTA Level 2 - Not Evidenced

2-CS-02, 2-NI-04, 2-NI-05, 2-NI-06, 2-DA-07, 2-DA-08, 2-DA-09, 2-AP-10, 2-AP-18, 2-IC-22

Recommended Level 2 Supplements

Standard(s)	Suggested supplement
2-CS-02	Add a hardware+software activity, such as micro:bit/Arduino sensor data displayed in an app.
2-NI-04, 2-NI-05, 2-NI-06	Add a mini-unit on protocols, passwords/security measures, and simple encryption models.
2-DA-07, 2-DA-08, 2-DA-09	Add a data mini-project using app logs, survey results, or API responses to encode, clean, visualize, and model data.
2-AP-10	Add planning pages requiring flowcharts or pseudocode before building each app.
2-AP-18, 2-IC-22	Add team roles, project timeline, peer review, and crowdsourced user feedback/survey components.

CSTA Level 3A - Evidenced

CSTA ID	Status	Where it's covered
3A-CS-02	Evidenced	Lessons 5, 7, 11, and 12 compare software layers, operating systems, native vs cross-platform tools, simulators/emulators, and app runtimes.
3A-CS-03	Evidenced	Lessons 1, 2, 4, 5, 12, 13, and 14 model troubleshooting through error copying, restarts, clean reloads, dependency checks, and device/build tests.
3A-AP-13	Evidenced	Lessons 1, 2, 3, 4, 6, and 12 create app prototypes in CodeSandbox, Snack, Bolt.new, Cursor, GitHub Codespaces, and Expo.
3A-AP-14	Partial	Lessons 2 and 3 use lists/arrays for laps, maze levels, suggestions, API results, and cached data; add explicit instruction on list generalization.
3A-AP-15	Partial	Lessons 2, 5, 6, 7, and 11 use conditionals, loops, intervals, and event handlers; add written justification for selected control structures.
3A-AP-16	Evidenced	Lessons 1, 2, 4, 9, and 11 develop interactive, event-driven artifacts using buttons, taps, movement controls, inputs, and UI feedback.
3A-AP-17	Evidenced	Lessons 3, 7, 9, 10, 11, and 12 decompose projects into components, screens, files, UI sections, assets, configuration, and publishing tasks.
3A-AP-18	Evidenced	Lessons 2, 5, 6, 7, and 11 combine procedures/functions, data/state, and UI behavior to create working computational artifacts.
3A-AP-19	Evidenced	Lessons 1, 2, 4, 11, 12, 13, and 14 require testing, refining, and preparing apps through preview panels, device tests, builds, and store testing.
3A-AP-21	Evidenced	Lessons 1, 8, 9, 10, and 12 refine usability, layout, responsiveness, screenshots, spacing, readability, and app behavior.
3A-AP-22	Partial	Lessons 1, 4, and 6 use accounts, forks, GitHub/Codespaces, and AI-assisted workflows; add formal team roles for full collaborative coverage.
3A-AP-23	Evidenced	Lessons 8, 9, 12, and 13 document design decisions through screenshots, UI specs, app metadata, release notes, README/API notes, and checklists.
3A-IC-24	Partial	Lessons 12, 13, and 14 address app permissions, privacy policies, store requirements, and publishing tradeoffs; add explicit impact analysis.
3A-IC-25	Partial	Lessons 3, 8, 9, and 10 support accessibility through touch targets, readability, responsiveness, and friendly error states; add explicit equity testing.

CSTA Level 3A - Not Evidenced

3A-CS-01, 3A-NI-04, 3A-NI-05, 3A-NI-06, 3A-NI-07, 3A-NI-08, 3A-DA-09, 3A-DA-10, 3A-DA-11, 3A-DA-12, 3A-AP-20, 3A-IC-26, 3A-IC-27, 3A-IC-28, 3A-IC-29, 3A-IC-30

Recommended Level 3A Supplements

Standard(s)	Suggested supplement
3A-CS-01, 3A-NI-04-08	Add explicit lessons on abstractions, internet protocols, network scalability/reliability, cybersecurity, and data protection.
3A-DA-09-12	Add a data project using API/survey/app data to collect, clean, visualize, model, and interpret results.
3A-AP-20	Add a short lesson on code/media licenses, attribution, open-source use, and acceptable reuse of AI-generated or copied code.
3A-IC-26-30	Add case studies on equity, access, bias, intellectual property, privacy, legal issues, and social impacts of computing.

CSTA Level 3B - Evidenced

CSTA ID	Status	Where it's covered
3B-CS-01	Evidenced	Lessons 5, 7, 11, and 12 compare Windows, macOS, iOS, Android, desktop, simulator/emulator, and cross-platform development environments.
3B-AP-16	Evidenced	Lessons 2, 3, 4, 9, and 12 use libraries, SDKs, APIs, packages, storage, Expo, React Native, and generated code structures.
3B-AP-17	Evidenced	Lessons 1-14 model the software lifecycle: design, code, test, revise, build, package, publish, and prepare store assets.
3B-AP-19	Evidenced	Lessons 2, 3, 4, 5, 7, 11, 12, 13, and 14 build or prepare apps for web, desktop, macOS, iOS, Android, and app stores.
3B-AP-20	Partial	Lessons 1, 4, and 6 use cloud tools, project folders, forks, GitHub/Codespaces, and IDE workflows; add formal version-control routines.
3B-AP-21	Evidenced	Lessons 12, 13, and 14 require smoke tests, device checks, APK/AAB/IPA builds, internal testing, TestFlight, and store submission review.
3B-AP-22	Evidenced	Lessons 4, 9, 12, 13, and 14 modify code, configurations, dependencies, assets, screenshots, metadata, and build settings to fix issues.
3B-AP-23	Partial	Lessons 10, 12, and 13 include code review prompts, QA checks, release notes, and publishing checklists; add peer code review for full coverage.
3B-AP-24	Evidenced	Lessons 1, 5, 7, 10, 11, and 12 compare frameworks, languages, native tools, cross-platform tools, no-code tools, and app targets.

CSTA Level 3B - Not Evidenced

3B-CS-02, 3B-NI-03, 3B-NI-04, 3B-DA-05, 3B-DA-06, 3B-DA-07, 3B-AP-08, 3B-AP-09, 3B-AP-10, 3B-AP-11, 3B-AP-12, 3B-AP-13, 3B-AP-14, 3B-AP-15, 3B-AP-18, 3B-IC-25, 3B-IC-26, 3B-IC-27, 3B-IC-28

Recommended Level 3B Supplements

Standard(s)	Suggested supplement
3B-CS-02, 3B-NI-03, 3B-NI-04	Add advanced operating system, network architecture, cybersecurity, and scalability/reliability lessons.
3B-DA-05, 3B-DA-06, 3B-DA-07	Add a larger data analytics unit with visualization, modeling, prediction, and evaluation of results.
3B-AP-08-15, 3B-AP-18	Add advanced algorithms, data structures, recursion, efficiency analysis, formal testing, and structured team project management.
3B-IC-25-28	Add debates/case studies on social impact, equity/access, laws/regulation, ethics, privacy, and responsible innovation.

Contents

1. Build App in CodeSandbox 1
   - First Project: Super Simple Calculator
   - Second Project: TI-10 Elementary Style Calculator
2. Making Apps on Snack Expo 9
   - Getting Started with Snack
   - App #1: Calculator
   - App #2: Stopwatch
   - App #3: Maze Game
3. Make an App using Bolt.new 61
   - Bolt.new Project Ideas and Framework Pathways
4. Build a React Expo App with Cursor 76
   - Create Expo Project
   - Use Cursor's AI to Generate the Calculator
   - Run the App on the Web
   - View App on your Phone using Expo Go
5. Make Your Own Windows App in VS Code 90
   - Install Visual Studio Code
   - Create a new project
   - Create Your Python Files
   - Package Your App as a Windows EXE
   - Why Python and VS Code Are Good Starting Points
6. Make App in GitHub Codespaces 102
   - Start GitHub Codespaces
   - More about GitHub
7. Make an App on Mac for Mac using Swift 115
   - Install Xcode
   - Replace your code with this setup
   - Run the App
   - Export as a Mac App
   - Create an App Icon
   - Generate Icon Sizes

Rebuild
Xcode and App Store Connect in the Apple Development Workflow

8. Design App User Interface in Figma 131
Lesson Goal
Create the iPhone Frame
Set Up Guides
Frame Background and Auto Layout
Calculator Body
Add Solar Panel
Display Screen
Counting Blocks – Visual Math Area
Buttons Grid Layout
Edit the buttons
Adjust Master Layout
Export Your Design
What Figma is used for and why it is popular

9. Convert Figma Design to Code 149
In Figma
Install Expo Go on your phone
Upload your PNG and paste the provided prompt
Preview your app on iOS, Android, and My Device
Start adding functionality to your code
User interface code and why design often comes first

10. Make User Interface using FlutterFlow 158
What You'll Make
Make the Deck Page
Make Page B
Review and Revise Code
Flutter and FlutterFlow

11. Make an App for iPhone on Mac using Storyboard 169
Run and view on phone simulator
Add the Swift Code
Run the App
Why some developers choose Storyboard instead of SwiftUI

12. Build a Mobile App with VS Code + Expo 192
Install Node.js – JavaScript runtime
Download or Open Visual Studio Code
Create your first Expo app

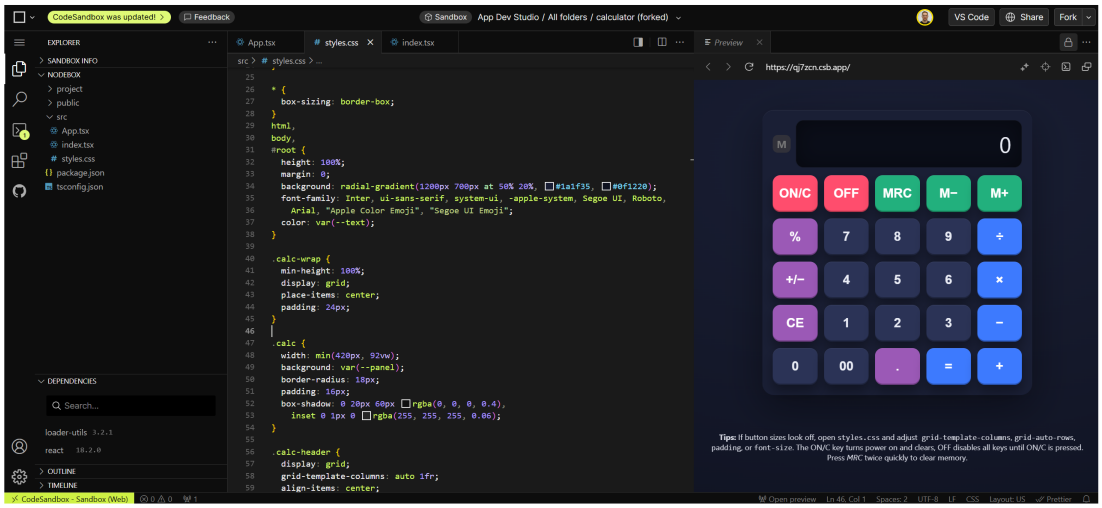
Code your App
Final configuration
Splash screen
Check your assets
Fix screens and navigation
Open it on a phone using Expo Go
Clean up dependencies
Perform a smoke test on your devices
Add a simple info/support screen
Match screenshots to the real app
Build the Android file (AAB)
Build the iOS file (IPA)
React Native, Expo, and Cross-Platform Publishing

13. Publish App on Google Play 216
Build the Android App
Install and Test the APK on Your Phone
Create the Google Play App Record
Begin Internal Testing
Setup internal testing
Google Play Console setup
App Screenshots
App Store Research and Positioning

14. Publish App on Apple App Store 236
Apple Developer Account
Install and Test on an iPhone
Prepare App For Submission

15. Resources and References 244
Code Editors and Development Environments
Web and Mobile App Development
Design Tools
Publishing Platforms
Additional Development Tools

1
Build App in CodeSandbox



In this lesson, you will build an app in CodeSandbox.io, an online development environment that lets you write and test code in your browser. You will create a new React project using the Vite + TypeScript template provided by CodeSandbox. After the project is created, you may use an external AI chat tool to generate code using the prompts provided in this lesson. You will then test your calculator in the preview panel and identify what needs improvement. Next, you will share that feedback with the AI tool to make revisions until your calculator matches your vision.

If you finish early, you are encouraged to explore the different coding templates available in CodeSandbox and learn why developers choose different frameworks for different projects. You can also try creating a simple project based on your own idea, using the tools and templates that best match what you want to build. If you encounter errors, paste them directly into the AI chat tool and follow its guidance. If you get stuck, you can ask the AI tool to rewrite the full code, or you can improve your prompt by describing your project more clearly so the AI tool can generate better results.

By the end of the lesson, you should have created a personalized calculator. Take a screenshot of your final result and share it with your teacher or class. If you create an additional project, take a screenshot of that project as well and share it too.

First Project: Super Simple Calculator

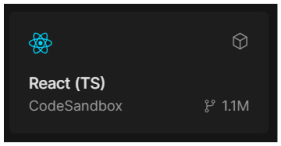
Open your browser and go to: https://codesandbox.io

Create an Account

- Click Sign In.
- Choose a sign-in option.
- Approve the connection so CodeSandbox can save your projects.

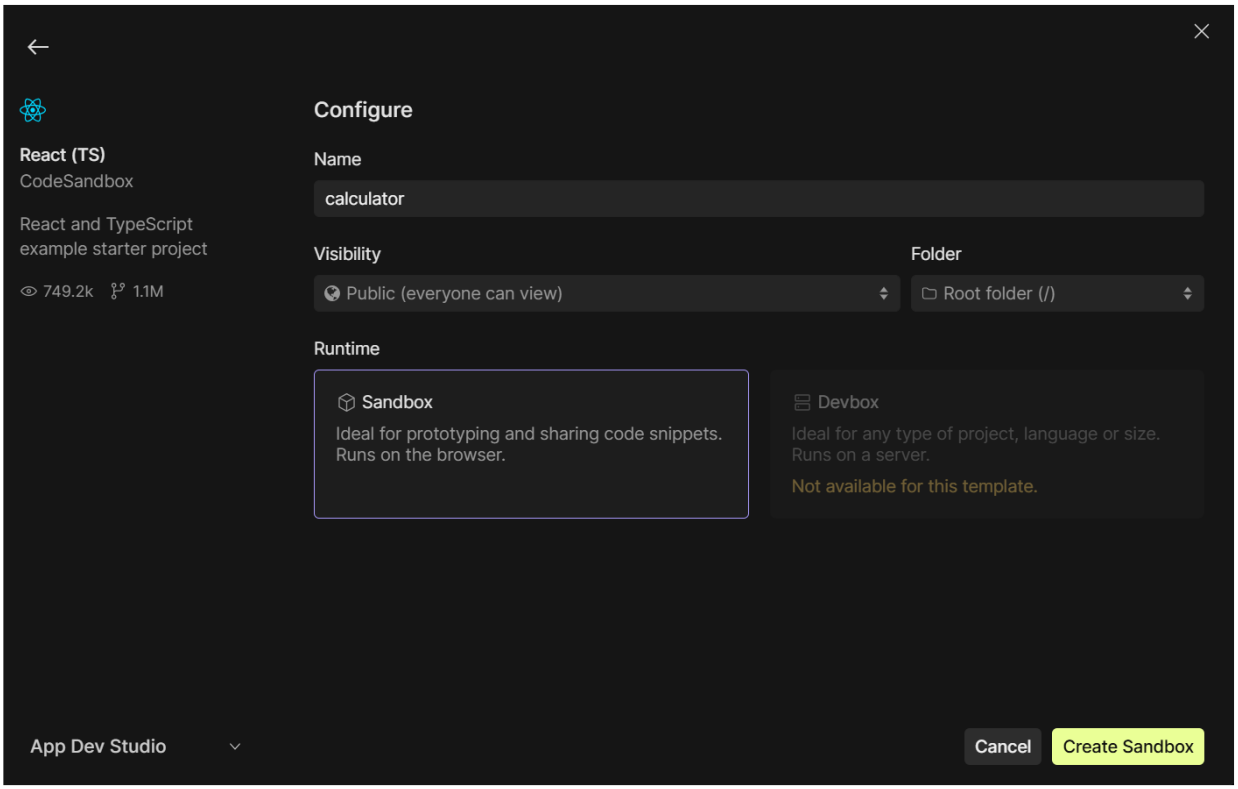
Start a New Project

- On the dashboard, click Create.
- Choose React (TS) as your template.

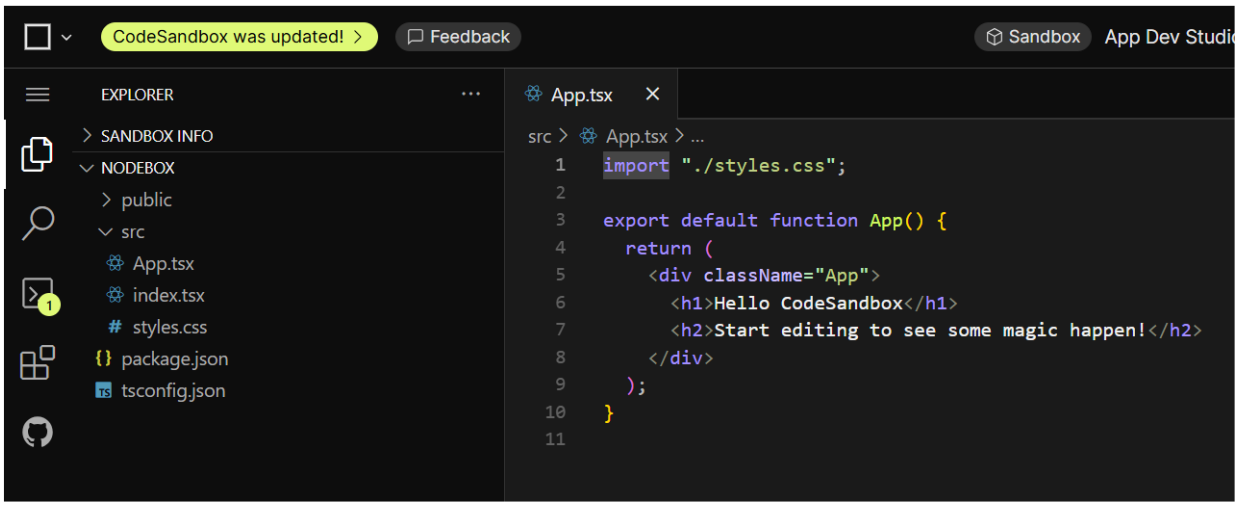


Choose Root Folder

- Name it calculator, set visibility according to your preference.
- Keep everything inside the project root folder (no need to create sub-folders).
- Click Create Sandbox.



By default, you'll see src/App.tsx, src/index.tsx, src/styles.css, package.json, and tsconfig.json.



Open an external AI chat such as GPT, DeepSeek, Claude, Gemini.

Use this prompt or similar.

Prompt:

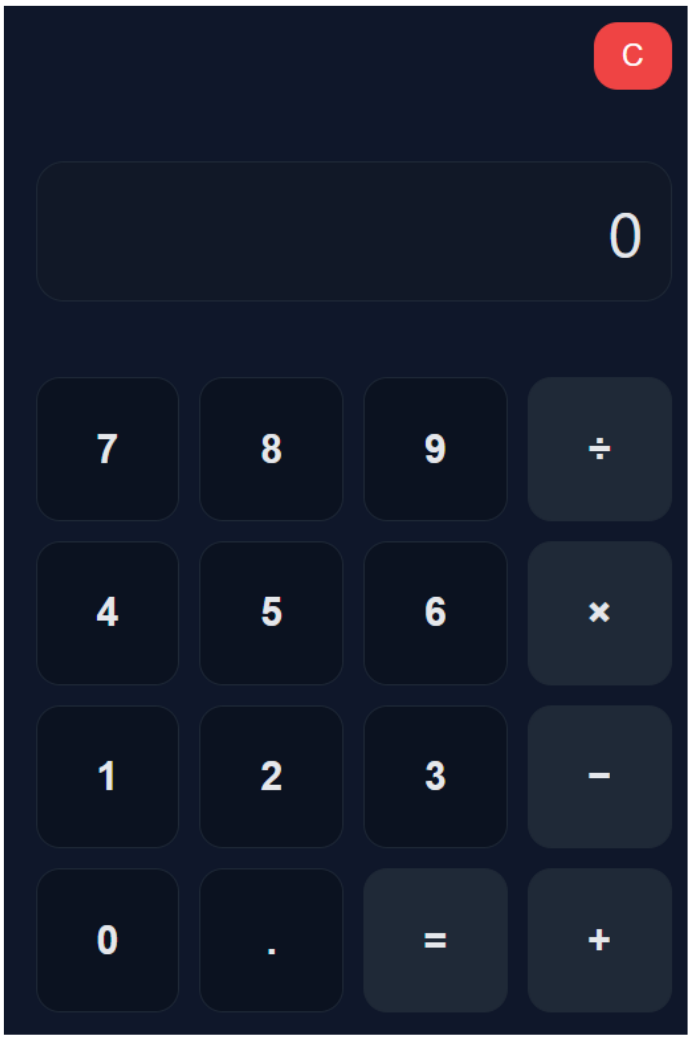
```
Create a super simple React (TS) calculator with these requirements:- One display at the top showing the current expression or 0.- A 4×4 grid of buttons with numbers 0–9 and the operators + − × ÷, plus C (clear), = (equals), and .- When a button is pressed, add it to the expression.- When = is pressed, evaluate safely and show the result.- Keep all code inside App.tsx.
```

Apply Changes

Copy and paste the code from your AI chat into your files control c and control v work fine. Make sure to replace all existing code inside App.tsx.

Test the Calculator

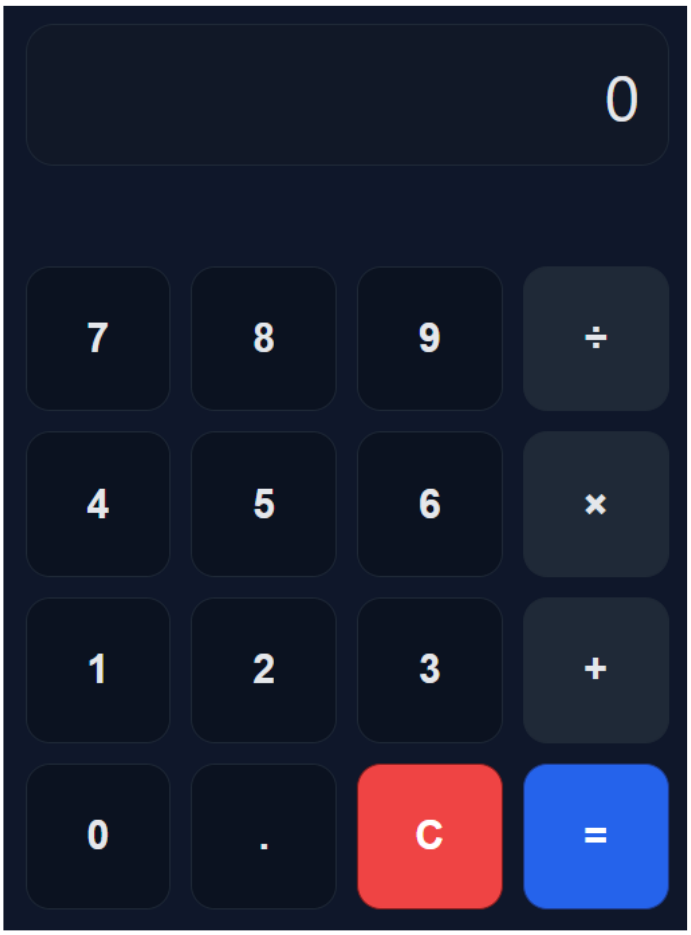
In the preview panel, try the different functions to see if they are working and check if you are satisfied with the design and layout. If it is not working or if you are not satisfied with the design and layout you may describe the issues to your AI agent. These buttons are in rows and columns so you could explain the issue according to this logic.



Here's the result I got. I'm not really satisfied with the look so I used this prompt to try and improve it.

Prompt:

```
Can we make the plus and minus button one single button so that we can move the
clear button to column 3 row 4. Rewrite the code for each file.
```



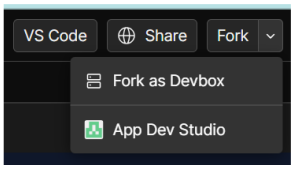
And here's the result I got. At this point if I wanted to adjust the space between the screen and the buttons I could ask the AI agent.

Prompt:

```
I think the screen is a little too far away from the buttons, can you tell
me which file and on what line of code I can change to adjust this?
```

Save Your Work Create a New Version

- Click Save at the top left of CodeSandbox, autosave was probably already enabled.
- To create a new version which will preserve the current version click Fork on the top right, Fork to the account you created.



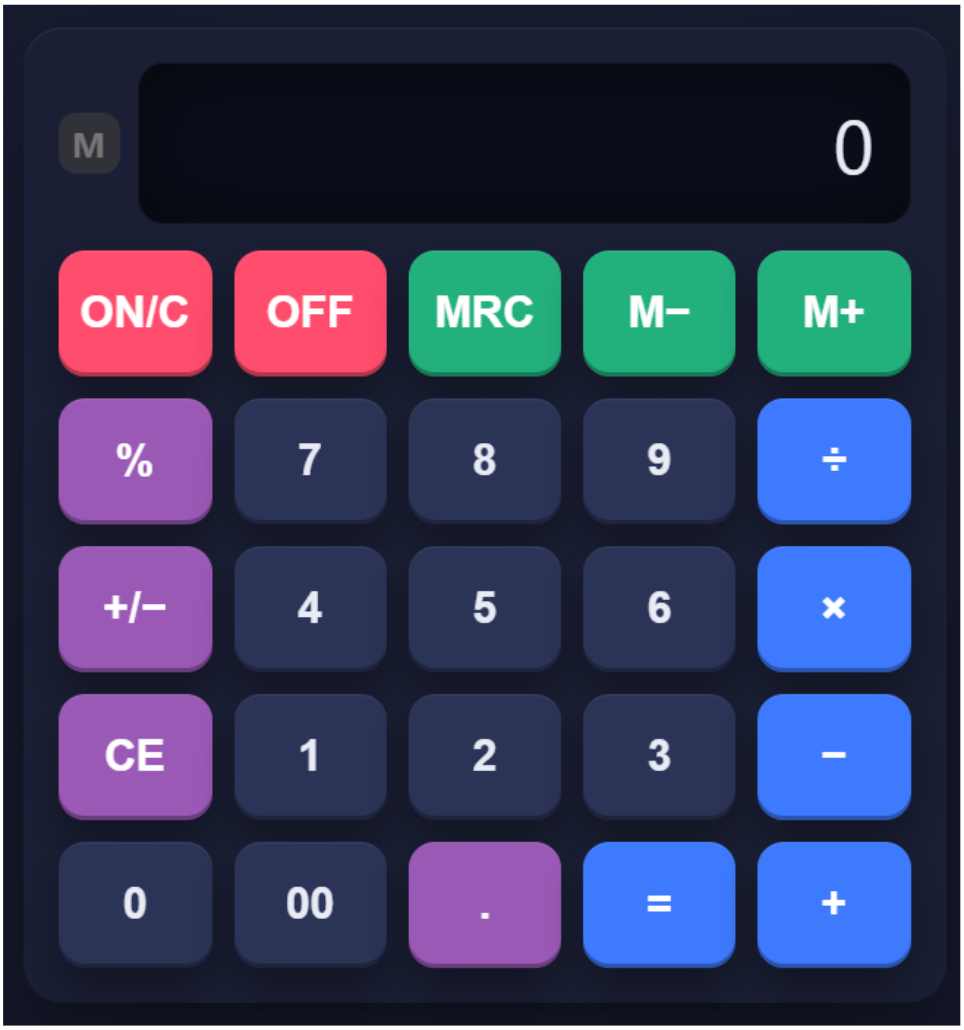
Second Project: TI-10 Elementary Style Calculator

Try this Prompt

```
Create a React (TS) calculator modeled after the TI-10 elementary calculator
layout:- 5 columns of buttons.- Row 1: ON/C, OFF, MRC, M−, M+.- Row 2: %, 7,
8, 9, ÷.- Row 3: +/−, 4, 5, 6, ×.- Row 4: CE, 1, 2, 3, −.- Row 5: 0, 00,
., =, +.Features:- ON/C turns the calculator on and clears everything.- OFF
disables the calculator until ON/C is pressed.- MRC recalls memory; pressing
twice quickly clears memory.- M+ and M− add or subtract the current value from
memory.- A memory indicator “M” should light up when memory is stored.- CE
clears only the current entry.- +/− toggles the sign of the current number.- %
divides the current number by 100.- = evaluates the expression safely and shows
the result.- Style the keys with large, colorful buttons (operators one color,
memory keys another, ON/C and OFF stand out).- Keep all code in App.tsx.
```

Apply and Test

Copy and paste the new code in the folders to replace the old code in App.tsx.



Here’s the result I got.

CodeSandbox is a great place to test different coding templates and frameworks quickly. The chart below can help you choose which template to explore based on what you want to build next. If you are working in a classroom, take a screenshot of your final calculator and share it with your teacher or class. If you finish early, start a new sandbox with a different template from the chart and use the same AI prompting process to build a simple project of your own. If you build a second project, take a screenshot of that one too.

Template	Category	Project Ideas	Strengths
HTML and CSS	Browser web page	A simple website page, a poster style page, or a class project page	Builds the layout and design of a page
JavaScript	Browser web app logic	A calculator, a quiz, a simple game, or interactive buttons and menus	Adds actions and logic such as clicks, math, timers, and saving simple data
Angular	Front end web app	A larger web app with multiple pages, a dashboard, or a bigger class project	Strong structure for bigger projects and many built in patterns
Next.js	Full stack web app	A real website with pages and simple server features	Lets you build the website and some server code in the same project
Node Express Server	Backend API server	A data server your app can call for users, scores, and saved results	Creates API routes such as /users or /scores that return data
React Native with Expo	Mobile app	A phone app such as flashcards, a habit tracker, or a simple game interface	Builds iOS and Android apps from one codebase

Chapter 1 Build an App in CodeSandbox Rubric

This chapter focused on creating a calculator app in CodeSandbox, testing its features, revising the design or functionality, and documenting the completed work.

Category	Points	Evidence of Completion
Project Setup	5	A CodeSandbox project is created using the correct React template. The project is clearly named, saved, and organized.
Calculator Build	5	The calculator includes a display, number buttons, operation buttons, a clear button, and an equals button.
App Functionality	5	Button presses respond correctly, input appears on the display, the clear button works, and basic calculations produce accurate results.
Testing and Revision	5	The calculator is tested with multiple examples. At least one issue, design concern, or improvement is identified and revised.
Evidence of Work	5	Screenshots and a CodeSandbox link, fork, or other project proof are submitted.
Reflection	5	A short explanation describes the completed app, the testing process, and relevant computer science vocabulary such as input, output, code, function, state, event, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
CodeSandbox project created	☐
Calculator app completed	☐
Calculator tested with at least three examples	☐
At least one improvement or revision made	☐
Screenshot of the final calculator included	☐
Project link or proof of work included	☐
Short reflection completed	☐

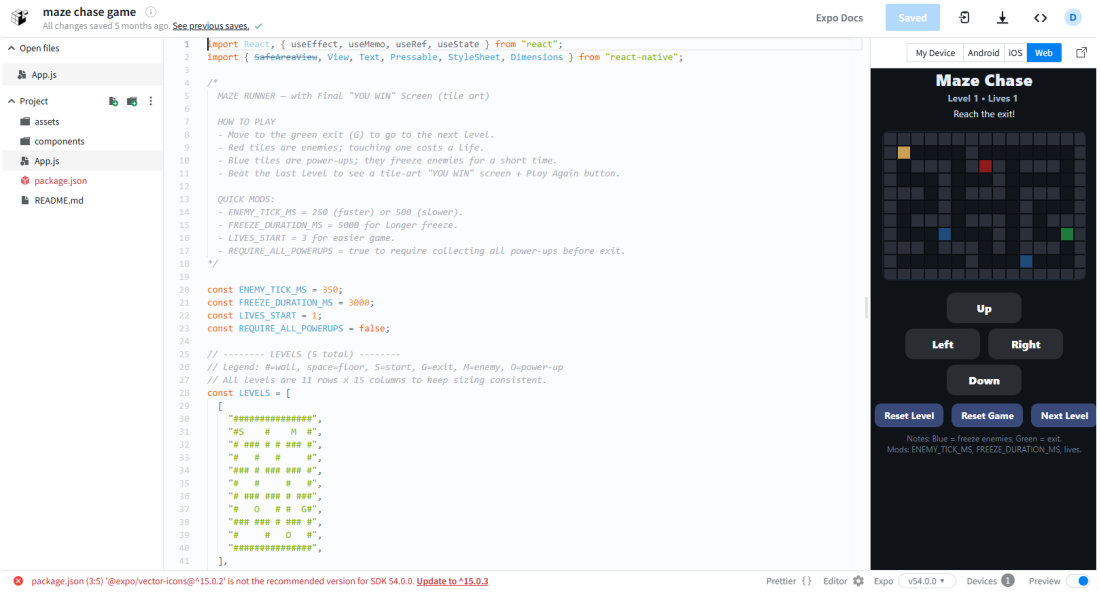
Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the calculator app created in CodeSandbox. Explain how the app uses input and output. Identify one part of the app that was tested, fixed, or improved before final submission.*

2

Making Apps on Snack Expo



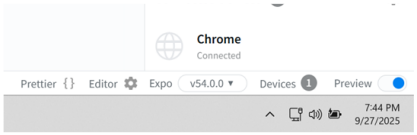
In this lesson you will learn how to use Snack at snack.expo.dev to build and test simple mobile apps directly in your browser. Snack lets you write React Native code, save your project online, and preview the results immediately. You will build three small apps, a calculator, a stopwatch, and a maze game, and you will use an AI assistant to help generate and revise the code. As you work, practice the core workflow you will use throughout this book. Open Snack, rename your project, replace the App.js code, save, and test in the preview panel and on your phone using Expo Go. Before you finish each app, update the Expo SDK version in Snack to the newest version so your project is more likely to run smoothly on your device.

By the end of this lesson you should have three working Snack projects and a clear process for building apps through small experiments and quick testing. If something breaks, copy the full error message and paste it into your AI assistant so it can help you fix the problem and regenerate the code. After each improvement, save and test again in both the preview panel and on your phone. When you are satisfied with each app, take a screenshot of your final results and share it as proof of completion. Most importantly, keep your projects saved in Snack so you can return later, reuse ideas, and build more advanced versions as you move into the next chapter.

Getting Started with Snack

- Open https://snack.expo.dev/ in your browser.
- Rename your project in the top left corner
- Replace all of App.js with the code provided in this guide.
- Save your project.
- Test the app in the preview panel or scan the QR code with Expo Go on your phone.

- At the bottom right corner of the screen change the Expo version to the newest version so that it will work on your phone later.



You can ask an AI assistant to generate code for you. Here are prompts you can use:

Calculator Prompt:

```
Build a simple React Native calculator for Snack (Expo). One screen. A display
at the top and a 4×4 grid of buttons. Include clear and equals buttons. Use
state. Convert ÷ × − to / * - before evaluating.
```

Stopwatch Prompt:

```
Create a React Native stopwatch for Snack (Expo) with Start, Pause, Reset, and
Lap. Show time as mm:ss:ms, update every 10ms. All code in App.js, no external
packages.
```

Maze Game Prompt:

```
Make a simple top-down maze game in React Native for Snack (Expo): render a
grid, player moves with on-screen buttons, enemies chase player, add powerups
and exit door to reach next level.
```

App #1: Calculator

AI Prompt:

```
Build a simple React Native calculator for Snack (Expo). One screen. A display
at the top and a 4×4 grid of buttons. Include clear and equals buttons. Use
state. Convert ÷ × − to / * - before evaluating.
```

Paste your own code or paste mine into snack.expo.dev inside App.js:

```
import React, { useState } from "react";

import { SafeAreaView, View, Text, Pressable, StyleSheet } from "react-native";



// These are the buttons shown in the calculator

//  You can add more symbols here, like "+" or "%"

//  You can also remove buttons you don't want

const BUTTONS = [
```

```
"7", "8", "9", "÷",
"4", "5", "6", "×",
"1", "2", "3", "−",
"C", "0", ".", "=",
];

// This changes pretty math symbols into real math signs JS can read
//  If you add more symbols, you may need to add replacements here
const symbolToJS = (s) =>
s.replace(/÷/g, "/").replace(/×/g, "*").replace(/−/g, "-");

// This checks if something is an operator (+, −, ×, ÷)
//  You can add more operators here if you add them to BUTTONS
const isOperator = (ch) => ["+", "−", "×", "÷"].includes(ch);

export default function App() {
// This keeps track of what is on the screen
//  Try changing "" to "123" to make it start with numbers already
const [display, setDisplay] = useState("");

// This runs whenever you press a button
const handlePress = (val) => {
if (val === "C") {
// Clear button → erase everything
//  You could change this to setDisplay("0") if you want it to show zero instead
setDisplay("");
return;
}
```

```
if (val === "=") {

// Equals button → solve the math

if (!display) return;



const expr = symbolToJS(display); // change symbols to JS operators



// Make sure only numbers and math signs are allowed

if (!/^[\d+\-*/.\s]+$/.test(expr)) {

setDisplay("Error"); // show Error if invalid

return;

}



// Do not allow ending with a math sign (like "7+")

if (/[+\-*/.]$/.test(expr)) {

setDisplay("Error");

return;

}



try {

// Do the math

const result = Function(` "use strict"; return (${expr}) ` )();



// Round off long decimals

//  Try changing 10 to 4 to round to only 4 decimal places

const shown =

typeof result === "number" && Number.isFinite(result)

? +parseFloat(result.toFixed(10))

: "Error";
```

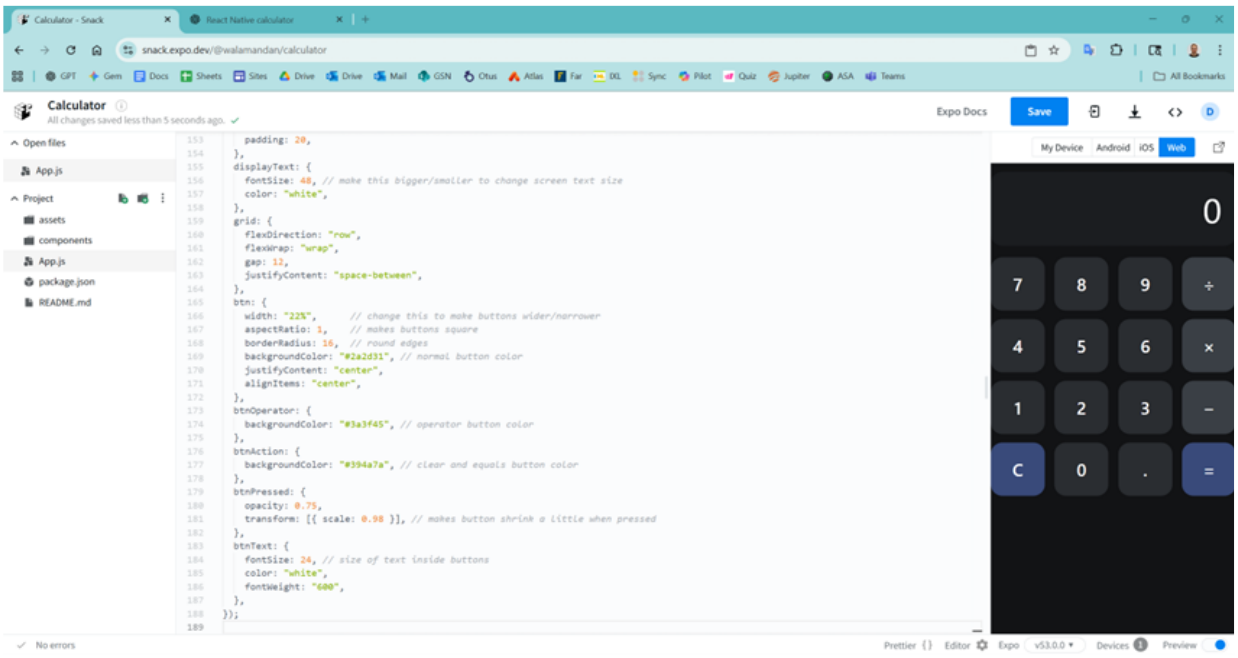
```
setDisplay(String(shown)); // show the answer

} catch (e) {

setDisplay("Error");

}

return;

}


// For numbers and operators, add them to the screen

setDisplay((prev) => {

let next = prev === "Error" ? "" : prev;


// Do not allow two operators in a row

//  Remove this if you want to allow things like "7--3"

if (isOperator(val)) {

if (next === "" && val !== "−") {

return next; // cannot start with + × ÷

}

if (next !== "" && isOperator(next.at(-1))) {

return next.slice(0, -1) + val; // replace last operator

}

}


// Do not allow more than one dot in the same number

//  Remove this rule if you want to test what happens

if (val === ".") {

const segs = next.split(/[+\−×÷]/);

const lastSeg = segs[segs.length - 1];

if (lastSeg.includes(".")) return next;

}
```

```
return next + val;
});
};

return (
<SafeAreaView style={styles.container}>
{/* This is the display at the top */}
{/*  Try changing "0" below to "Ready" so it says Ready at the start */}
<View style={styles.display}>
<Text style={styles.displayText} numberOfLines={1}>
{display || "0"}
</Text>
</View>

{/* This makes the 4×4 grid of buttons */}
{/*  Try adding more buttons to BUTTONS to see how it looks */}
<View style={styles.grid}>
{BUTTONS.map((b) => (
<CalcButton key={b} label={b} onPress={() => handlePress(b)} />
))}
</View>
</SafeAreaView>
);
}

// This makes one button
//  Try changing font size or colors in styles.btnText to see new looks
function CalcButton({ label, onPress }) {
```

```
const isAction = label === "C" || label === "="; // "C" and "=" buttons

const isOp = ["÷", "×", "−", "+"].includes(label); // operator buttons


return (

<Pressable

onPress={onPress}

style={({ pressed }) => [

styles.btn,

isAction && styles.btnAction,

isOp && styles.btnOperator,

pressed && styles.btnPressed, // small effect when pressed

]}

>

<Text style={styles.btnText}>{label}</Text>

</Pressable>

);

}


// This is the style section

//  Change numbers/colors here to change how the app looks

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: "#121315", // background of the whole app

paddingHorizontal: 16,

paddingBottom: 20,

//  Try changing backgroundColor to "white" for light mode

},

display: {
```

```
marginTop: 12,

marginBottom: 16,

minHeight: 96, // height of display area

borderRadius: 16,

backgroundColor: "#1b1d20", // display background

justifyContent: "flex-end",

alignItems: "flex-end",

padding: 20,

//  Try alignItems: "flex-start" to move numbers to the left

},

displayText: {

fontSize: 48, // text size on display

color: "white",

//  Try making this 60 to make the numbers bigger

//  Try changing color to "black" if using light background

},

grid: {

flexDirection: "row",

flexWrap: "wrap",

gap: 12,

justifyContent: "space-between",

},

btn: {

width: "22%",    // how wide each button is

aspectRatio: 1,   // keeps buttons square

borderRadius: 16, // round corners

backgroundColor: "#2a2d31", // button background color

justifyContent: "center",

alignItems: "center",
```

```
// Try changing width to "30%" to make fewer but bigger buttons per row

},

btnOperator: {

backgroundColor: "#3a3f45", // operator buttons color

// Try making this "#ff9800" for orange operator buttons

},

btnAction: {

backgroundColor: "#394a7a", // clear and equals color

// Try changing this to "red" for clear, "green" for equals

},

btnPressed: {

opacity: 0.75, // makes it look faded when pressed

transform: [{ scale: 0.98 }], // makes button shrink a little when pressed

// Try changing scale to 0.9 for a bigger "press" effect

},

btnText: {

fontSize: 24, // size of numbers on buttons

color: "white",

fontWeight: "600",

// Try changing color to "yellow" to make the button text stand out

},

});
```



Try This

Now look at the code and read the notes, try slightly modifying the code to see what kind of changes you can get. For example you can change the numbers to letters, but if you try to use the calculator this way you'll get an error, so you will want to change them back after you try.

```
// List of all buttons that will show up
// 👉 You can add more (like "+", "%") or remove some here
const BUTTONS = [
  "G", "H", "I", "÷",
  "D", "E", "F", "×",
  "A", "B", "C", "−",
  "C", "0", ".", "=",
];
```

App #2: Stopwatch

AI Prompt:

`Create a React Native stopwatch for Snack (Expo) with Start, Pause, Reset, and Lap. Show time as mm:ss:ms, update every 10ms. All code in App.js, no external packages.`

Paste your own code or paste mine into snack.expo.dev inside App.js:

import React, { useEffect, useRef, useState } from "react";

import { SafeAreaView, View, Text, Pressable, FlatList, StyleSheet } from "react-native";

/*

```
STOPWATCH — Simple, clear notes + mod ideas

WHAT IT DOES

- Shows time in mm:ss:cs (minutes:seconds:centiseconds)

- Updates every 10 ms when running

- Start: begins counting

- Pause: stops counting but keeps the time

- Reset: sets time back to 0 and clears laps

- Lap: saves the current time to a list

MOD IDEAS (TRY THESE!)

- Change UPDATE_MS to 50 for lower power use (smoother at 10, cheaper at 50)

- Change styles (colors, sizes) to make your own theme

- Show hours if you time very long events (see formatTime)

- Disable “Lap” when not running (already done below)

- Show both "Lap time" and "Split time" (already shown)

*/


const UPDATE_MS = 10; // how often the display updates (10ms = centiseconds)

/*  MOD: Try 20 or 50 to reduce CPU usage; 5 to be extra smooth (costs more battery) */


// Formats milliseconds → "mm:ss:cs"

function formatTime(ms) {

//  We store time in milliseconds, but show centiseconds (cs = ms/10)

const totalCs = Math.floor(ms / 10);

const minutes = Math.floor(totalCs / 6000);          // 6000 cs = 60 seconds

const seconds = Math.floor((totalCs % 6000) / 100);    // 100 cs = 1 second

const cs = totalCs % 100;
```

```
const pad2 = (n) => String(n).padStart(2, "0");

return `${pad2(minutes)}:${pad2(seconds)}:${pad2(cs)}`;


/* MOD (show hours):

const hours = Math.floor(minutes / 60);

const m = minutes % 60;

return `${pad2(hours)}:${pad2(m)}:${pad2(seconds)}:${pad2(cs)}`;

*/

}


export default function App() {

// elapsed: total time in ms

const [elapsed, setElapsed] = useState(0);

// running: is the stopwatch active?

const [running, setRunning] = useState(false);

// laps: list of elapsed times (ms) when Lap was pressed

const [laps, setLaps] = useState([]);


// Refs so values persist without re-rendering

const intervalRef = useRef(null);  // holds setInterval id

const startRef = useRef(0);       // time when we last started (Date.now() - alreadyElapsed)


// Start the stopwatch

const handleStart = () => {

if (running) return; // already running

setRunning(true);


// We anchor the start time so there's no drift: now - already elapsed

startRef.current = Date.now() - elapsed;
```

```
// Update elapsed every UPDATE_MS
intervalRef.current = setInterval(() => {
setElapsed(Date.now() - startRef.current);
}, UPDATE_MS);
};

// Pause the stopwatch
const handlePause = () => {
if (!running) return;
setRunning(false);
if (intervalRef.current) {
clearInterval(intervalRef.current);
intervalRef.current = null;
}
// elapsed already holds the latest value from the interval
};

// Reset everything
const handleReset = () => {
setRunning(false);
if (intervalRef.current) {
clearInterval(intervalRef.current);
intervalRef.current = null;
}
setElapsed(0);
setLaps([]);
//  MOD: Keep laps on reset? Remove setLaps([]) to keep them
};
```

```
// Save a lap (split) time
const handleLap = () => {
if (!running) return; // only lap when running
// Add newest lap to the top
setLaps((prev) => [elapsed, ...prev]);

/*  MOD: Put newest lap at the bottom instead:
setLaps((prev) => [...prev, elapsed]);
*/
};

// Clean up the interval if the component unmounts
useEffect(() => {
return () => {
if (intervalRef.current) clearInterval(intervalRef.current);
};
}, []);

// Compute lap durations vs previous lap (so we can show both “Lap” and “Split”)
// split = time at lap; lapTime = time difference since prior lap (or since 0)
const getLapRows = () => {
return laps.map((splitMs, index) => {
const prevSplit = laps[index + 1] ?? 0;      // next item in array is the previous lap in time
const lapMs = splitMs - prevSplit;          // duration of this lap only
return {
key: String(index),
index: laps.length - index,               // Lap #1 is oldest; show human-friendly numbering
lapTime: formatTime(lapMs),
```

```
splitTime: formatTime(splitMs),
};
});
};

const toggleStartPause = running ? handlePause : handleStart;

return (
<SafeAreaView style={styles.container}>
{/* DISPLAY — shows the main stopwatch time */}
{/*  MOD: Change fontSize in styles.timeText to 80 for huge time */}
<View style={styles.display}>
<Text style={styles.timeText}>{formatTime(elapsed)}</Text>
</View>

{/* BUTTONS — Start/Pause, Lap, Reset */}
{/*  MOD: Swap the order or add icons by changing these three Pressables */}
<View style={styles.controlsRow}>
<ActionButton
label={running ? "Pause" : "Start"}
onPress={toggleStartPause}
type={running ? "pause" : "start"}
/>
<ActionButton
label="Lap"
onPress={handleLap}
disabled={!running} // lap only when running
type="lap"
/>
```

```
<ActionButton
label="Reset"
onPress={handleReset}
disabled={running && elapsed > 0 ? false : elapsed === 0} // disable if already zero
type="reset"
/>
</View>

{/* LAPS LIST — shows Lap #, Lap time, and Split time */}
{/*  MOD: Remove Split column if you only want Lap durations */}
<View style={styles.lapsHeader}>
<Text style={[styles.lapsHeaderText, { flex: 1 }]}>#</Text>
<Text style={[styles.lapsHeaderText, { flex: 3 }]}>Lap</Text>
<Text style={[styles.lapsHeaderText, { flex: 3 }]}>Split</Text>
</View>

<FlatList
data={getLapRows()}
keyExtractor={(item) => item.key}
contentContainerStyle={{ paddingBottom: 24 }}
renderItem={({ item }) => (
<View style={styles.lapRow}>
<Text style={[styles.lapIndex, { flex: 1 }]}>{item.index}</Text>
<Text style={[styles.lapText, { flex: 3 }]}>{item.lapTime}</Text>
<Text style={[styles.lapText, { flex: 3 }]}>{item.splitTime}</Text>
</View>
)}
ListEmptyComponent={
<Text style={styles.emptyText}>
```

```
No laps yet. Press Start, then Lap.

</Text>

}

/>

</SafeAreaView>

);

}


/* Small button component so styles are consistent

 MOD: Change type-based colors below to fit your theme

*/

function ActionButton({ label, onPress, disabled, type = "default" }) {

const base = [styles.btn];

if (type === "start") base.push(styles.btnStart);

if (type === "pause") base.push(styles.btnPause);

if (type === "lap") base.push(styles.btnLap);

if (type === "reset") base.push(styles.btnReset);

if (disabled) base.push(styles.btnDisabled);


return (

<Pressable onPress={onPress} disabled={disabled} style={({ pressed }) => [

...base,

pressed && !disabled && styles.btnPressed,

]}>

<Text style={styles.btnText}>{label}</Text>

</Pressable>

);

}
```

```
// ---------- STYLES ----------
//  MOD: Change colors for light theme (e.g., container bg to "#ffffff", text to "#111")
const styles = StyleSheet.create({
container: {
flex: 1,
backgroundColor: "#0f1216",
padding: 16,
},
display: {
backgroundColor: "#151a20",
borderRadius: 16,
paddingVertical: 24,
paddingHorizontal: 20,
alignItems: "center",
justifyContent: "center",
marginBottom: 16,
},
timeText: {
fontSize: 56, //  Try 72 for extra large time
fontWeight: "700",
color: "#e8f0fe",
letterSpacing: 1,
},
controlsRow: {
flexDirection: "row",
gap: 12,
marginBottom: 12,
},
btn: {
```

```
flex: 1,

paddingVertical: 14, //  Try 18 to make buttons taller

borderRadius: 12,

alignItems: "center",

justifyContent: "center",

},

btnStart: { backgroundColor: "#2e7d32" },

btnPause: { backgroundColor: "#b26a00" },

btnLap:   { backgroundColor: "#394a7a" },

btnReset: { backgroundColor: "#8b1c1c" },

btnDisabled: { opacity: 0.5 },

btnPressed: { transform: [{ scale: 0.98 }], opacity: 0.85 },

btnText: {

color: "white",

fontSize: 18,

fontWeight: "700",

},

lapsHeader: {

flexDirection: "row",

paddingHorizontal: 6,

paddingVertical: 8,

borderBottomWidth: StyleSheet.hairlineWidth,

borderBottomColor: "#2b3340",

},

lapsHeaderText: {

color: "#9fb3c8",

fontWeight: "700",

fontSize: 14,

textTransform: "uppercase",
```

```
letterSpacing: 0.5,

},

lapRow: {

flexDirection: "row",

paddingVertical: 10,

paddingHorizontal: 6,

borderBottomWidth: StyleSheet.hairlineWidth,

borderBottomColor: "#1f2530",

},

lapIndex: { color: "#b7c6d9", fontWeight: "700", fontSize: 16 },

lapText:  { color: "#e0e8f5", fontSize: 16, fontVariant: ["tabular-nums"] },

emptyText: {

color: "#6f7e91",

textAlign: "center",

paddingVertical: 16,

},

});
```

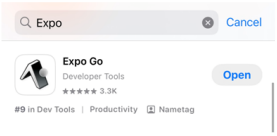
Try This

- Ask your AI agent this: Can you add countdown mode to the stopwatch?
- Paste the updated code into snack.expo.dev

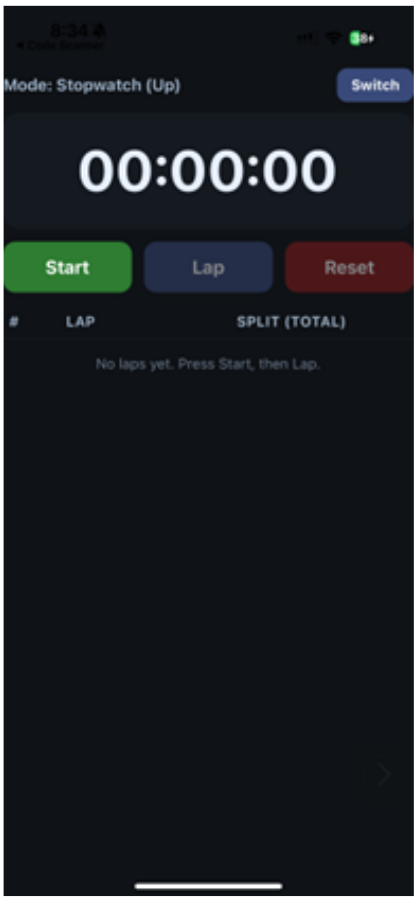
Try using the app on your phone

Change the view to My Device

Download Expo Go from the app store



Use your camera to scan the code inside snack.expo.dev to use your app on your phone

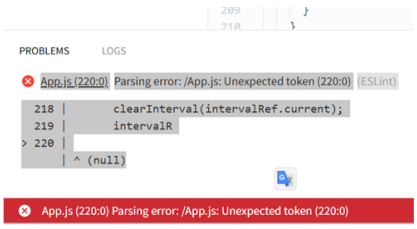


App #3: Maze Game

AI Prompt:

```
Make a simple top-down maze game in React Native for Snack (Expo): render a
grid, player moves with on-screen buttons, enemies chase player, add powerups
and exit door to reach next level.
```

Note: If you get an error while coding



Copy your error code and paste it into your AI agent chat along with the prompt:

AI Prompt:

`Can you fix this error and remake the code?`

The AI agent fixed the code and returned the edited version below.

Paste your own code or paste mine into snack.expo.dev inside App.js:

```
import React, { useEffect, useMemo, useRef, useState } from "react";

import { SafeAreaView, View, Text, Pressable, StyleSheet, Dimensions } from "react-native";


/*

SIMPLE MAZE GAME (fixed version)

- Player moves with on-screen buttons

- Enemies chase player

- Powerups freeze enemies

- Reach exit to complete level

*/


// ====== SETTINGS ======

const ENEMY_TICK_MS = 350;

const FREEZE_DURATION_MS = 3000;

const LIVES_START = 1;

const REQUIRE_ALL_POWERUPS = false;


// ====== LEVELS ======

const LEVELS = [

[

"###############",

"#S  #    M  #",

"# ### ### ### #",

"#  #  #    #",

"### # ### ### #",

"#  #    #  #",
```

```
"# ### ### # ###",
"#   O   # #  G#",
"### ### # ### #",
"#    #   O   #",
"################",
],
];

const dirs = {
up:   { r: -1, c:  0 },
down:  { r:  1, c:  0 },
left:  { r:  0, c: -1 },
right: { r:  0, c:  1 },
};

function inBounds(grid, r, c) {
return r >= 0 && r < grid.length && c >= 0 && c < grid[0].length;
}

function isWall(grid, r, c) {
return grid[r][c] === "#";
}

function parseLevel(levelRows) {
const grid = levelRows.map((row) => row.split(""));
let player = { r: 0, c: 0 };
const enemies = [];
const powerups = new Set();
let exit = { r: 0, c: 0 };
```

```
grid.forEach((row, r) => {
row.forEach((ch, c) => {
if (ch === "S") {
player = { r, c };
grid[r][c] = " ";
} else if (ch === "M") {
enemies.push({ r, c });
grid[r][c] = " ";
} else if (ch === "O") {
powerups.add(` ${r},${c}` );
grid[r][c] = " ";
} else if (ch === "G") {
exit = { r, c };
grid[r][c] = " ";
}
});
});
return { grid, player, enemies, powerups, exit };
}


function nextEnemyStep(grid, from, target) {
const options = [
{ dr: -1, dc: 0 },
{ dr: 1, dc: 0 },
{ dr: 0, dc: -1 },
{ dr: 0, dc: 1 },
];
options.sort((a, b) => {
```

```
const distA = Math.abs(from.r + a.dr - target.r) + Math.abs(from.c + a.dc - target.c);

const distB = Math.abs(from.r + b.dr - target.r) + Math.abs(from.c + b.dc - target.c);

return distA - distB;

});

for (const m of options) {

const nr = from.r + m.dr;

const nc = from.c + m.dc;

if (inBounds(grid, nr, nc) && !isWall(grid, nr, nc)) {

return { r: nr, c: nc };

}

}

return { ...from };

}



export default function App() {

const [levelIndex, setLevelIndex] = useState(0);

const { grid: baseGrid, player: startPlayer, enemies: startEnemies, powerups: startPowerups, exit } =

useMemo(() => parseLevel(LEVELS[levelIndex % LEVELS.length]), [levelIndex]);



const [grid] = useState(baseGrid);

const [player, setPlayer] = useState(startPlayer);

const [enemies, setEnemies] = useState(startEnemies);

const [powerups, setPowerups] = useState(startPowerups);

const [lives, setLives] = useState(LIVES_START);

const [frozenUntil, setFrozenUntil] = useState(0);

const [message, setMessage] = useState("Reach the exit!");

const [won, setWon] = useState(false);

const [lost, setLost] = useState(false);
```

```
const cols = grid[0].length;

const rows = grid.length;

const { width } = Dimensions.get("window");

const padding = 16;

const cellSize = Math.floor((width - padding * 2) / cols);



const tryMove = (dr, dc) => {

if (won || lost) return;

const nr = player.r + dr;

const nc = player.c + dc;

if (!inBounds(grid, nr, nc) || isWall(grid, nr, nc)) return;



const posKey = `${nr},${nc}`;

if (powerups.has(posKey)) {

setPowerups((prev) => {

const copy = new Set(prev);

copy.delete(posKey);

return copy;

});

setFrozenUntil(Date.now() + FREEZE_DURATION_MS);

setMessage("Enemies frozen!");

}



setPlayer({ r: nr, c: nc });

if (nr === exit.r && nc === exit.c) {

if (REQUIRE_ALL_POWERUPS && powerups.size > 0) {

setMessage("Collect all power-ups first!");

} else {

setWon(true);
```

```
setMessage("Level complete!");

}

}

};



const intervalRef = useRef(null);

useEffect(() => {

if (intervalRef.current) clearInterval(intervalRef.current);

intervalRef.current = setInterval(() => {

if (won || lost) return;

const now = Date.now();

const frozen = now < frozenUntil;

if (frozen) return;



setEnemies((prev) => {

const next = prev.map((e) => nextEnemyStep(grid, e, player));

if (next.some((e) => e.r === player.r && e.c === player.c)) {

setLives((L) => {

const left = L - 1;

if (left <= 0) {

setLost(true);

setMessage("Game Over!");

} else {

setPlayer(startPlayer);

setMessage(` Ouch! Lives left: ${left}` );

}

return Math.max(0, left);

});

}
```

```
return next;

});

}, ENEMY_TICK_MS);

return () => {

if (intervalRef.current) clearInterval(intervalRef.current);

};

}, [grid, player, won, lost, frozenUntil, startPlayer]);


const renderCell = (r, c) => {

const isPlayer = player.r === r && player.c === c;

const isEnemy = enemies.some((e) => e.r === r && e.c === c);

const isPower = powerups.has(` ${r},${c}`);

const isExit = exit.r === r && exit.c === c;

const wall = isWall(grid, r, c);


let bg = wall ? "#2b2f3a" : "#11161d";

if (isExit) bg = "#1e7a3a";

if (isPower) bg = "#1e4b7a";

if (isEnemy) bg = "#8b1c1c";

if (isPlayer) bg = "#caa052";


return (

<View

key={` ${r}-${c}` }

style={{

width: cellSize,

height: cellSize,

backgroundColor: bg,

borderWidth: 1,
```

```
borderColor: "#0b0f14",
}}
/>
);
};

return (
<SafeAreaView style={styles.container}>
<Text style={styles.title}>Maze Runner</Text>
<Text style={styles.subtitle}>
Level {levelIndex + 1} • Lives {lives}
</Text>
<Text style={styles.message}>{message}</Text>

<View style={[styles.grid, { width: cellSize * cols, height: cellSize * rows }]}>
{grid.map((row, r) => (
<View key={`row-${r}`} style={{ flexDirection: "row" }}>
{row.map((_, c) => renderCell(r, c))}
</View>
))}
</View>

<View style={styles.controls}>
<Pressable style={styles.btn} onPress={() => tryMove(dirs.up.r, dirs.up.c)}><Text style={styles.btnText}>Up</Text></Pressable>
<View style={{ flexDirection: "row" }}>
<Pressable style={styles.btn} onPress={() => tryMove(dirs.left.r, dirs.left.c)}><Text style={styles.btnText}>Left</Text></Pressable>
<Pressable style={styles.btn} onPress={() => tryMove(dirs.right.r, dirs.right.c)}><Text style={styles.btnText}>Right</Text></Pressable>
</View>
<Pressable style={styles.btn} onPress={() => tryMove(dirs.down.r, dirs.down.c)}><Text style={styles.btnText}>Down</Text></Pressable>
```

```
</View>

</SafeAreaView>

);

}



const styles = StyleSheet.create({

container: { flex: 1, backgroundColor: "#0f1216", alignItems: "center", padding: 12 },

title: { color: "#e8f0fe", fontSize: 24, fontWeight: "800" },

subtitle: { color: "#9fb3c8", marginTop: 2, fontWeight: "700" },

message: { color: "#dbe7ff", marginTop: 4 },

grid: { marginVertical: 10 },

controls: { alignItems: "center", marginTop: 10 },

btn: { backgroundColor: "#2a2d31", padding: 12, margin: 4, borderRadius: 8 },

btnText: { color: "white", fontWeight: "700" },

});
```

Try This

In order to restart your game you will need to press refresh or change the view between Web and iOS, To fix this you can ask your AI agent to add a game reset button to your game.

Here's my prompt:

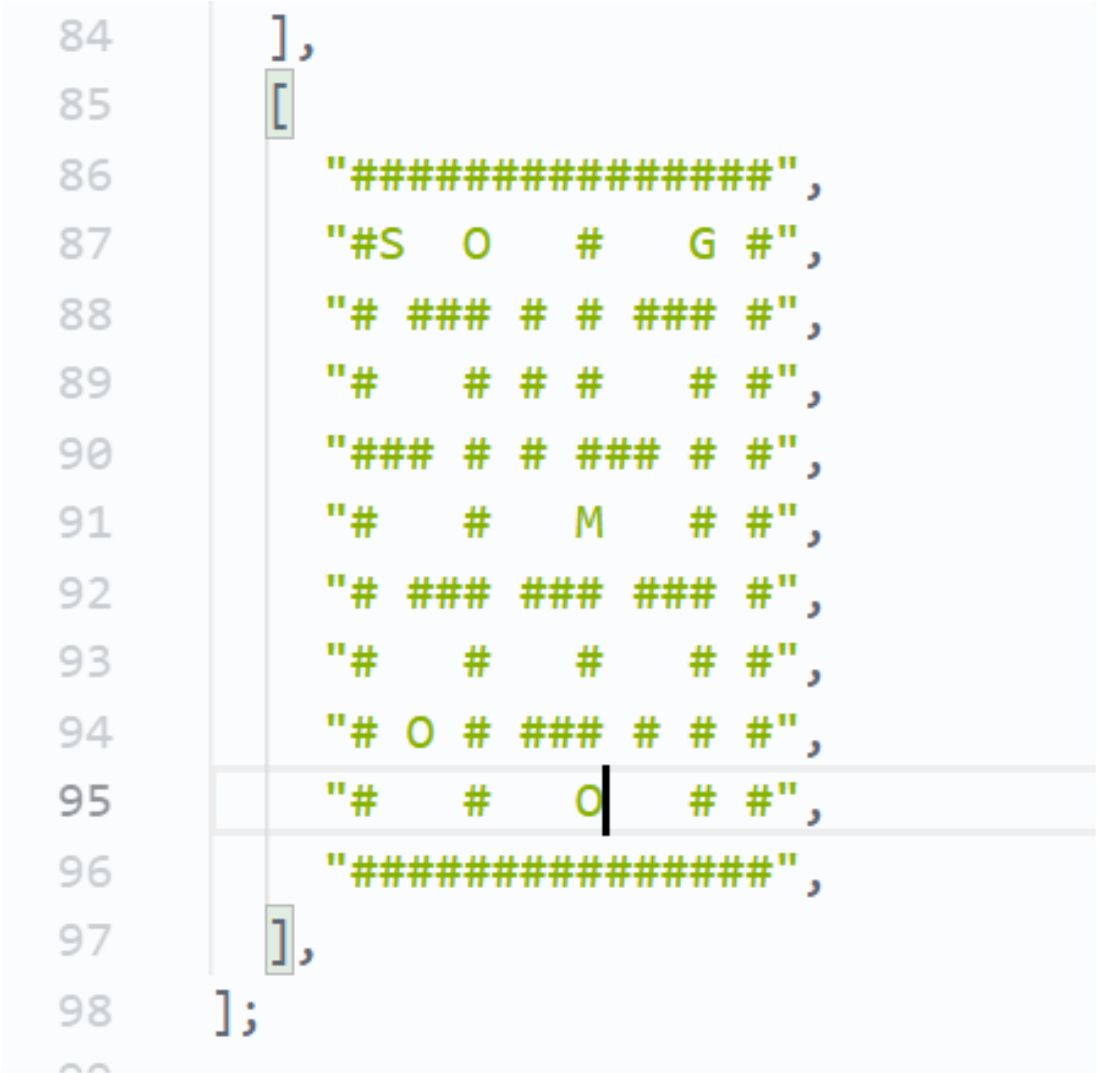
```
Add a reset button to the game, create an additional 4 levels.
```

Each time test your game to check for improvements, on level 3 in my game it was too difficult to win, so I prompted the AI chat to change the layout of level 3.

Remember to test your app on your phone and fix or improve any errors you discover.

You can update your levels manually too:

Legend: #=wall, space=floor, S=start, G=exit, M=enemy, O=power-up



After making some updates to the levels, here's my final version for this tutorial:

import React, { useEffect, useMemo, useRef, useState } from "react";

import { SafeAreaView, View, Text, Pressable, StyleSheet, Dimensions } from "react-native";

/*

MAZE RUNNER — with Final "YOU WIN" Screen (tile art)

HOW TO PLAY

- Move to the green exit (G) to go to the next level.

- Red tiles are enemies; touching one costs a life.

- Blue tiles are power-ups; they freeze enemies for a short time.

- Beat the last level to see a tile-art "YOU WIN" screen + Play Again button.

QUICK MODS:

- ENEMY_TICK_MS = 250 (faster) or 500 (slower).

- FREEZE_DURATION_MS = 5000 for longer freeze.

- LIVES_START = 3 for easier game.

- REQUIRE_ALL_POWERUPS = true to require collecting all power-ups before exit.

*/

```
const ENEMY_TICK_MS = 350;

const FREEZE_DURATION_MS = 3000;

const LIVES_START = 1;

const REQUIRE_ALL_POWERUPS = false;


// -------- LEVELS (5 total) --------

// Legend: #=wall, space=floor, S=start, G=exit, M=enemy, O=power-up

// All levels are 11 rows x 15 columns to keep sizing consistent.

const LEVELS = [

[

"###############",

"#S   #   M  #",

"# ### # # ### #",

"#  #  #   #",

"### # ### ### #",

"#  #   #  #",

"# ### ### # ###",

"#  O  # #  G#",

"### ### # ### #",

"#   #  O  #",
```

```
"################",
],
[
"################",
"#S      O   M#",
"# ### ### ### #",
"#  #    #  #",
"# ### ### ### #",
"#  #  O  # #",
"### # ##### # #",
"#  #  #  # #",
"# ### # # ### #",
"#  O  #  MG #",
"################",
],
// Newer Level 3 layout (wider corridors, two power-ups)
[
"################",
"#S #    #  #",
"# # ### # # # #",
"# #  # #  # #",
"# ### # ### # #",
"#  O  #  # #",
"### ### # ### #",
"#  #  #  # #",
"# # # ### #M# #",
"#M#  O  # G #",
"################",
],
```

```
[
"################",
"#S   #   MO #",
"# ### # ### # #",
"#  ##  ### #",
"### # # # # # #",
"# O #  # # # #",
"# # ### # # # #",
"# #  # # # # #",
"# ### # # # # #",
"#  M  #  G #",
"################",
],
[
"################",
"#S O  #  G #",
"# ### # # ### #",
"#  ### ##",
"### # # ### # #",
"#  #  M  # #",
"# ### ### ### #",
"#  #  #  # #",
"# O # ### # # #",
"#  #  O  M#G#",
"################",
],
];

// Directions for D-pad
```

```
const dirs = {
up:   { r: -1, c:  0 },
down: { r:  1, c:  0 },
left: { r:  0, c: -1 },
right: { r:  0, c:  1 },
};

// ---------- helpers ----------
function inBounds(grid, r, c) {
return r >= 0 && r < grid.length && c >= 0 && c < grid[0].length;
}
function isWall(grid, r, c) {
return grid[r][c] === "#";
}
function parseLevel(levelRows) {
// Read ASCII layout → grid + entities
const grid = levelRows.map((row) => row.split(""));
let player = { r: 0, c: 0 };
const enemies = [];
const powerups = new Set();
let exit = { r: 0, c: 0 };

grid.forEach((row, r) => {
row.forEach((ch, c) => {
if (ch === "S") { player = { r, c }; grid[r][c] = " "; }
else if (ch === "M") { enemies.push({ r, c }); grid[r][c] = " "; }
else if (ch === "O") { powerups.add(`${r},${c}`); grid[r][c] = " "; }
else if (ch === "G") { exit = { r, c }; grid[r][c] = " "; }
});
```

```
});

return { grid, player, enemies, powerups, exit };

}


// Simple greedy enemy step (easy to follow; swap with BFS if you want)

function nextEnemyStep(grid, from, target) {

const moves = [

{ dr: -1, dc: 0 }, { dr: 1, dc: 0 }, { dr: 0, dc: -1 }, { dr: 0, dc: 1 },

].sort((a, b) => {

const da = Math.abs((from.r + a.dr) - target.r) + Math.abs((from.c + a.dc) - target.c);

const db = Math.abs((from.r + b.dr) - target.r) + Math.abs((from.c + b.dc) - target.c);

return da - db;

});

for (const m of moves) {

const nr = from.r + m.dr, nc = from.c + m.dc;

if (inBounds(grid, nr, nc) && !isWall(grid, nr, nc)) return { r: nr, c: nc };

}

return { ...from };

}


// ---------- WIN TILE ART ----------

// 11 rows x 15 columns of " " (off) and "X" (lit).

// It's a trophy + "WIN" made from the same square cells.

const WIN_ART = [

"    XXX      ",

"   XXXXX     ",

"  XXXXXXX    ",

"  XX XXX     ",

"  XXXXXXX    ",
```

```
"   XXXXX  X ",
" X XXX  X  ",
" X      X ",
" XXXX X XX ",
" X X XXX X ",
"           ",
];
// Colors for the win art (you can tweak for fun)
const WIN_ON_COLOR = "#caa052"; // gold (same as player)
const WIN_OFF_COLOR = "#11161d"; // normal floor

export default function App() {
const [levelIndex, setLevelIndex] = useState(0);

// Parse current level when index changes
const parsed = useMemo(
() => parseLevel(LEVELS[(levelIndex % LEVELS.length + LEVELS.length) % LEVELS.length]),
[levelIndex]
);

// Per-level static grid
const [grid, setGrid] = useState(parsed.grid);
// Dynamic state
const [player, setPlayer] = useState(parsed.player);
const [enemies, setEnemies] = useState(parsed.enemies);
const [powerups, setPowerups] = useState(parsed.powerups);
const [exit, setExit] = useState(parsed.exit);
const [lives, setLives] = useState(LIVES_START);
const [frozenUntil, setFrozenUntil] = useState(0);
```

```
const [won, setWon] = useState(false);     // per-level win flash

const [lost, setLost] = useState(false);

const [message, setMessage] = useState("Reach the exit!");

const [gameWon, setGameWon] = useState(false); // final “YOU WIN” state


// When level changes, (re)load its state (but keep lives, gameWon stays as-is unless we reset the game)

useEffect(() => {

setGrid(parsed.grid);

setPlayer(parsed.player);

setEnemies(parsed.enemies);

setPowerups(parsed.powerups);

setExit(parsed.exit);

setFrozenUntil(0);

setWon(false);

setLost(false);

setMessage("Reach the exit!");

}, [parsed]);


// Grid sizing

const cols = grid[0].length;

const rows = grid.length;

const { width } = Dimensions.get("window");

const padding = 16;

const cellSize = Math.floor((width - padding * 2) / cols);


// Try to move player

const tryMove = (dr, dc) => {

if (won || lost || gameWon) return;

const nr = player.r + dr, nc = player.c + dc;
```

```
if (!inBounds(grid, nr, nc) || isWall(grid, nr, nc)) return;


const posKey = `${nr},${nc}`;
if (powerups.has(posKey)) {
// Pick up → freeze enemies
setPowerups((prev) => {
const copy = new Set(prev);
copy.delete(posKey);
return copy;
});
setFrozenUntil(Date.now() + FREEZE_DURATION_MS);
setMessage("Enemies frozen!");
}


setPlayer({ r: nr, c: nc });


// Exit check
if (nr === exit.r && nc === exit.c) {
if (REQUIRE_ALL_POWERUPS && powerups.size > 0) {
setMessage("Collect all power-ups first!");
} else {
// If last level → final win screen; else go to next level
const lastLevel = (levelIndex % LEVELS.length) === LEVELS.length - 1;
if (lastLevel) {
setGameWon(true);
setWon(true);
setMessage("You Win!");
} else {
setWon(true);
```

```
setMessage("Level complete!");
setTimeout(() => setLevelIndex((i) => i + 1), 600);
}
}
}
};

// Enemy loop
const intervalRef = useRef(null);
useEffect(() => {
if (intervalRef.current) clearInterval(intervalRef.current);
intervalRef.current = setInterval(() => {
if (won || lost || gameWon) return; // stop when level won, game over, or final win screen
const now = Date.now();
if (now < frozenUntil) return; // frozen → skip

setEnemies((prev) => {
const next = prev.map((e) => nextEnemyStep(grid, e, player));
// collision?
if (next.some((e) => e.r === player.r && e.c === player.c)) {
setLives((L) => {
const left = L - 1;
if (left <= 0) {
setLost(true);
setMessage("Game Over!");
} else {
setPlayer(parsed.player); // respawn at start
setMessage(` Ouch! Lives left: ${left}`);
}
```

```
return Math.max(0, left);

});

}

return next;

});

}, ENEMY_TICK_MS);



return () => { if (intervalRef.current) clearInterval(intervalRef.current); };

}, [grid, player, won, lost, gameWon, frozenUntil, parsed.player]);



// Reset Level (keeps lives, stays on same level)

const resetLevel = () => {

if (gameWon) return;

setPlayer(parsed.player);

setEnemies(parsed.enemies);

setPowerups(parsed.powerups);

setFrozenUntil(0);

setWon(false);

setLost(false);

setMessage("Level reset!");

};



// Reset Game (back to level 1, restore lives, clear final win screen)

const resetGame = () => {

setLevelIndex(0);

setLives(LIVES_START);

setFrozenUntil(0);

setWon(false);

setLost(false);
```

```
setGameWon(false);

setMessage("New game!");

// Level state refreshes via effect on `parsed`

};



// Render a single cell for normal game

const renderCell = (r, c) => {

const isPlayer = player.r === r && player.c === c;

const isEnemy  = enemies.some((e) => e.r === r && e.c === c);

const isPower  = powerups.has(`${r},${c}`);

const isExit   = exit.r === r && exit.c === c;

const wall     = isWall(grid, r, c);



let bg = wall ? "#2b2f3a" : "#11161d"; // wall vs floor

if (isExit)   bg = "#1e7a3a";        // exit = green

if (isPower)  bg = "#1e4b7a";        // power-up = blue

if (isEnemy)  bg = "#8b1c1c";        // enemy = red

if (isPlayer) bg = "#caa052";        // player = gold



return (

<View

key={`${r}-${c}`}

style={{

width: cellSize, height: cellSize,

backgroundColor: bg,

borderWidth: 1, borderColor: "#0b0f14",

}}

/>

);
```

```
};

// -------- FINAL WIN SCREEN (tile art) --------
// Draws WIN_ART using the same square tiles/colors.
const WinGrid = () => {
// Ensure art matches grid dimensions (11x15). If not, center it on the area.
const artRows = WIN_ART.length;      // 11
const artCols = WIN_ART[0].length;   // 15
const padTop = Math.max(0, Math.floor((rows - artRows) / 2));
const padLeft = Math.max(0, Math.floor((cols - artCols) / 2));

// Simple pulse effect: alternate gold shades every ~400ms
const [pulse, setPulse] = useState(false);
useEffect(() => {
const t = setInterval(() => setPulse((p) => !p), 400);
return () => clearInterval(t);
}, []);

const onColor = pulse ? "#d6b568" : WIN_ON_COLOR;

const cells = [];
for (let r = 0; r < rows; r++) {
const rowCells = [];
for (let c = 0; c < cols; c++) {
const ar = r - padTop;
const ac = c - padLeft;
const lit = (ar >= 0 && ar < artRows && ac >= 0 && ac < artCols && WIN_ART[ar][ac] === "X");
rowCells.push(
<View
```

```
key={` w-${r}-${c}` }
style={{
width: cellSize,
height: cellSize,
backgroundColor: lit ? onColor : WIN_OFF_COLOR,
borderWidth: 1,
borderColor: "#0b0f14",
}}
/>
);
}
cells.push(
<View key={` wr-${r}` } style={{ flexDirection: "row" }}>
{rowCells}
</View>
);
}
return (
<View style={[styles.grid, { width: cellSize * cols, height: cellSize * rows }]}>
{cells}
</View>
);
};

const levelNum = ((levelIndex % LEVELS.length) + LEVELS.length) % LEVELS.length + 1;

return (
<SafeAreaView style={styles.container}>
{/* HEADER */}
```

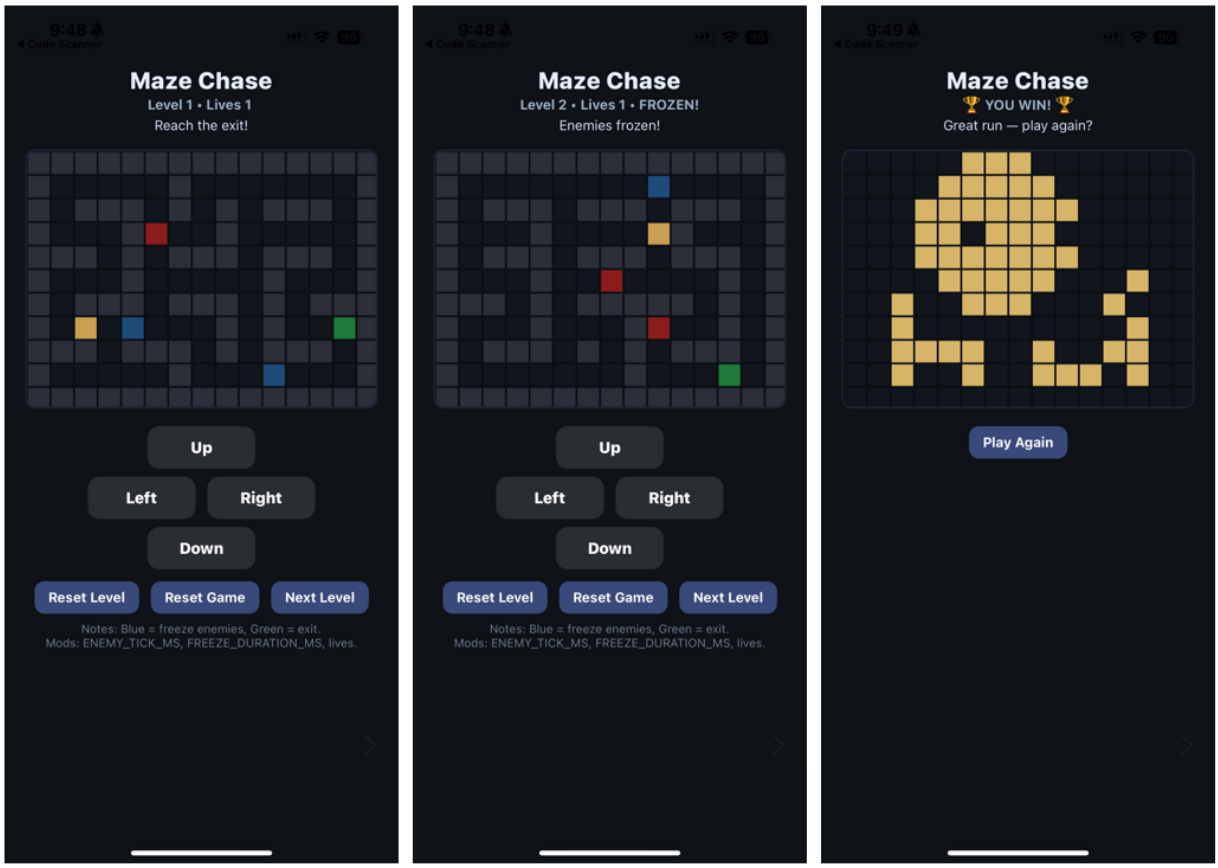
```
<View style={styles.header}>
<Text style={styles.title}>Maze Chase</Text>
{!gameWon ? (
<>
<Text style={styles.subtitle}>
Level {levelNum} • Lives {lives} {Date.now() < frozenUntil ? "• FROZEN!" : ""}
</Text>
<Text style={styles.message}>{message}</Text>
</>
) : (
<>
<Text style={styles.subtitle}> YOU WIN! </Text>
<Text style={styles.message}>Great run — play again?</Text>
</>
)}
</View>

{/* MAIN: either game grid or final win tile-art */}
{!gameWon ? (
<View style={[styles.grid, { width: cellSize * cols, height: cellSize * rows }]}>
{grid.map((row, r) => (
<View key={ `row-${r}` } style={{ flexDirection: "row" }}>
{row.map((_, c) => renderCell(r, c))}
</View>
))}
</View>
) : (
<WinGrid />
)}
```

```
{/* CONTROLS / ACTIONS */}

{!gameWon ? (

<>

{/* D-PAD */}

<View style={styles.controls}>

<View style={styles.row}>

<PadButton label="Up" onPress={() => tryMove(dirs.up.r, dirs.up.c)} />

</View>

<View style={styles.row}>

<PadButton label="Left" onPress={() => tryMove(dirs.left.r, dirs.left.c)} />

<PadButton label="Right" onPress={() => tryMove(dirs.right.r, dirs.right.c)} />

</View>

<View style={styles.row}>

<PadButton label="Down" onPress={() => tryMove(dirs.down.r, dirs.down.c)} />

</View>

</View>


{/* FOOTER ACTIONS */}

<View style={styles.footer}>

<SmallButton label="Reset Level" onPress={() => resetLevel()} />

<SmallButton label="Reset Game" onPress={() => resetGame()} />

<SmallButton label="Next Level" onPress={() => setLevelIndex((i) => i + 1)} />

</View>

</>

) : (

<View style={styles.footer}>

<SmallButton label="Play Again" onPress={resetGame} />

</View>
```

```
)}

{/* QUICK HELP */}
{!gameWon && (
<View style={styles.help}>
<Text style={styles.helpText}>Notes: Blue = freeze enemies, Green = exit.</Text>
<Text style={styles.helpText}>Mods: ENEMY_TICK_MS, FREEZE_DURATION_MS, lives.</Text>
</View>
)}
</SafeAreaView>
);
}

// ---- small UI components ----
const PadButton = ({ label, onPress }) => (
<Pressable onPress={onPress} style={({ pressed }) => [styles.btn, pressed && styles.pressed]}>
<Text style={styles.btnText}>{label}</Text>
</Pressable>
);
const SmallButton = ({ label, onPress }) => (
<Pressable onPress={onPress} style={({ pressed }) => [styles.smallBtn, pressed && styles.pressed]}>
<Text style={styles.smallBtnText}>{label}</Text>
</Pressable>
);

// ---- styles ----
const styles = StyleSheet.create({
container: { flex: 1, backgroundColor: "#0f1216", alignItems: "center", padding: 12 },
header: { alignItems: "center", marginBottom: 6 },
```

```
title: { color: "#e8f0fe", fontSize: 24, fontWeight: "800", letterSpacing: 0.5 },

subtitle: { color: "#9fb3c8", marginTop: 2, fontWeight: "700" },

message: { color: "#dbe7ff", marginTop: 4 },


grid: {

backgroundColor: "#0b0f14",

borderWidth: 2,

borderColor: "#1b2531",

borderRadius: 10,

overflow: "hidden",

marginVertical: 10,

},


controls: { marginTop: 4, alignItems: "center" },

row: { flexDirection: "row", gap: 12, marginVertical: 4 },


btn: {

backgroundColor: "#2a2d31",

paddingVertical: 12,

paddingHorizontal: 24,

borderRadius: 12,

minWidth: 110,

alignItems: "center",

},

btnText: { color: "white", fontWeight: "800", fontSize: 16 },


smallBtn: {

backgroundColor: "#394a7a",

paddingVertical: 8,
```

```
paddingHorizontal: 14,

borderRadius: 10,

marginHorizontal: 6,

},

smallBtnText: { color: "white", fontWeight: "700" },

pressed: { opacity: 0.85, transform: [{ scale: 0.98 }] },


footer: { flexDirection: "row", marginTop: 8 },


help: { marginTop: 8, alignItems: "center" },

helpText: { color: "#6f7e91", fontSize: 12 },

});
```



Many students are especially interested in making mobile apps, so this chapter uses React Native with Expo to help you build apps faster and test them on a phone. The data in the table below shows that platform specific tools still dominate their own stores. Swift appears in about 83% of App Store apps, and Kotlin appears in about 79% of Google Play apps.

At the same time, the table also shows why cross platform development matters. After the top native option in each store, several of the next most common development tools are cross platform, including Flutter, React Native, Cordova, and Expo. For example, on the App Store, Flutter and React Native appear more often than the next platform specific item in the list, MetalKit. On Google Play, Flutter and React Native are also among the most common options after Kotlin. These percentages describe how many apps include each SDK, and apps can include more than one, so the values are best read as a popularity comparison rather than parts of a single total.

React Native supports iOS and Android apps using JavaScript and native platform capabilities, which means you can share most of your code across both platforms while still writing platform specific code when you need it.

Most Popular Mobile App SDKs

Apple App Store Apps		Google Play Store Apps	
Swift	83%	Kotlin	79%
Flutter	16%	Flutter	26%
React Native	14%	Jetpack Compose	22%
Cordova	9%	React Native	19%
Metal Kit	7%	Cordova	10%
Expo	7%	Expo	8%

Source: Appfigures, The most popular development SDKs [1]

Chapter 2 Making Apps on Snack Expo Rubric

This chapter focused on creating mobile app projects in Snack Expo, testing them in the preview panel or on a device, revising errors, and documenting completed work.

Category	Points	Evidence of Completion
Project Setup	5	Snack Expo projects are created, renamed, saved, and set to an appropriate Expo SDK version.
App Builds	5	The required app projects are created, including a calculator, stopwatch, and maze game or approved alternatives.
App Functionality	5	The apps respond to user interaction. Calculator buttons work, stopwatch controls function, and the maze game includes movement and basic gameplay.
Testing and Revision	5	Each app is tested in the preview panel or on a device. Errors, layout problems, or gameplay issues are identified and revised.
Evidence of Work	5	Screenshots, Snack project links, QR code testing proof, or other project evidence are submitted.
Reflection	5	A short explanation describes the completed apps, the testing process, and relevant computer science vocabulary such as input, output, state, event, variable, function, loop, condition, debugging, or user interface.
Total	**30**	

Submission Checklist

Requirement	Check
Snack Expo project or projects created	☐
Project names updated and saved	☐
Expo SDK version checked or updated	☐
Calculator app completed	☐
Stopwatch app completed	☐
Maze game completed or approved alternative completed	☐
Apps tested in preview panel or on a device	☐
At least one error, layout issue, or gameplay issue revised	☐
Screenshots or project links included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

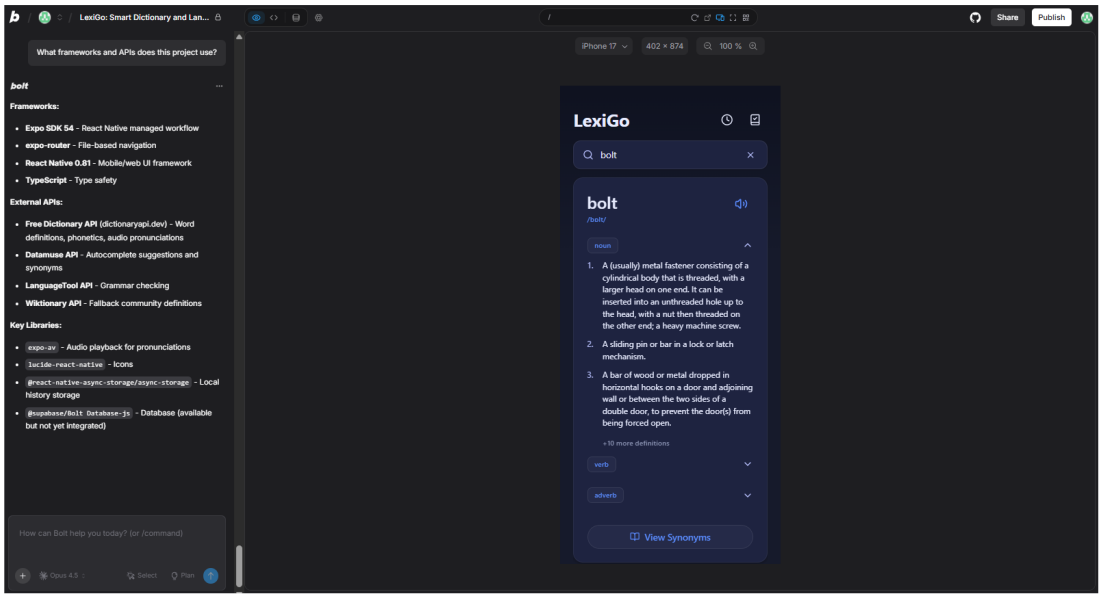
> *Describe the mobile apps created in Snack Expo. Explain how one app uses state, events, or user input. Identify one error, design issue, or gameplay problem that was tested and improved before final submission.*

---

1. Appfigures, “The Most Popular Development SDKs” (Type: Development; Category: Apps), Appfigures, accessed February 20, 2026, https://appfigures.com/top-sdks/development/apps

3

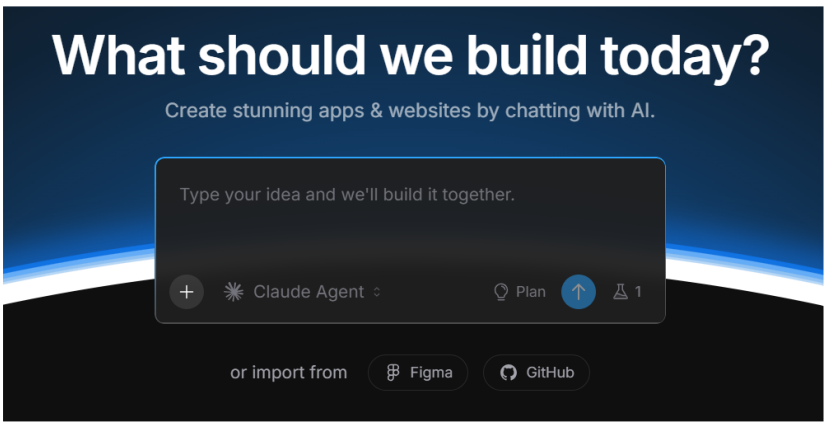
Make an App using Bolt.new


LexiGo: Smart Dictionary and Lan...
Share
Publish
What frameworks and APIs does this project use?
bolt
Frameworks:
Expo SDK 54 - React Native managed workflow
expo-router - File-based navigation
React Native 0.81 - Mobile/web UI framework
TypeScript - Type safety
External APIs:
Free Dictionary API (dictionaryapi.dev) - Word definitions, phonetics, audio pronunciations
Datamuse API - Autocomplete suggestions and synonyms
LanguageTool API - Grammar checking
Wiktionary API - Fallback community definitions
Key Libraries:
expo-av - Audio playback for pronunciations
lucide-react-native - Icons
@react-native-async-storage/async-storage - Local history storage
@supabase/Bolt Database-js - Database (available but not yet integrated)
How can Bolt help you today? (or /command)
iPhone 17
402 × 874
100 %
LexiGo
bolt
bolt
/bolt/
noun
1. A (usually) metal fastener consisting of a cylindrical body that is threaded, with a larger head on one end. It can be inserted into an unthreaded hole up to the head, with a nut then threaded on the other end; a heavy machine screw.
2. A sliding pin or bar in a lock or latch mechanism.
3. A bar of wood or metal dropped in horizontal hooks on a door and adjoining wall or between the two sides of a double door, to prevent the door(s) from being forced open.
+10 more definitions
verb
adverb
View Synonyms

In this lesson you will learn how to use Bolt.new as an AI powered coding workspace for building a React Native app with Expo. You will start a new project, write a clear prompt that describes what you want to build, and then review the files Bolt generates so you can understand how the app is structured. You will practice an important real world workflow used by developers, which is to build in small steps, test often, and improve the code based on what you see in the preview. You will also learn how to run the app on your own phone using Expo Go by scanning a QR code, and you will finish by publishing a web version so others can access your app quickly.

Open Bolt.new

- Go to https://bolt.new in your web browser.
- Sign in with your account (GitHub preferred).



In the chat box, type a prompt like this:

```
Make a simple calculator app with React Native and Expo.
```

You can also create a more complex setup by working with your AI chatbot to plan the app in advance, but be cautious because longer prompts can lead to more errors or incomplete code. Some developers prefer to start by generating only the file structure or only the user interface, then ask Bolt.new to build the UI elements and add functionality one feature at a time. In this lesson, I will paste the full prompt below to create a dictionary app. This prompt was created by developing a more comprehensive app plan in another AI chat outside of Bolt.new, and if you want to try this approach, you can copy it into your own AI chat and replace the details to fit your own app plan.

Prompt:

- Create a React Native + Expo (TypeScript) app called LexiGo: a fast, modern dictionary + thesaurus with spell-help and grammar check. It must run on iOS, Android, and Web (Expo Web) and only call no-auth/no-key endpoints
- Definitions + phonetics: https://api.dictionaryapi.dev/api/v2/entries/en/<word>
- Thesaurus + suggestions: https://api.datamuse.com (/sug, ?rel_syn=, ?ml=, etc.)
- Grammar/spell check: https://api.languagetool.org/v2/check (public endpoint)
- Fallback (extract text): Wiktionary via MediaWiki API

User stories (must have):

- As a user, I can type a word and immediately see smart suggestions (“did you mean…”) driven by Datamuse /sug, debounced.
- When I submit a word, I see a Definition Card with:
- Headword, phonetic(s), play audio button (if audio exists)
- Part(s) of speech with short, readable definitions and example sentences
- A subtle Merriam-style look: clean card, generous spacing, tags
- I also see a Thesaurus section with synonym chips (Datamuse rel_syn). Tapping a chip runs a new lookup.
- I can open a Grammar & Spell Check panel (LanguageTool) to paste a sentence/paragraph and view inline suggestions with one-tap replacement.
- If DictionaryAPI.dev 404s or is sparse, show a Wiktionary fallback extract (clearly labeled “Community definitions”).
- The app is fast and respectful of rate limits:
- Debounce search (300-400ms)
- Cache recent results in memory and AsyncStorage

- Batch LanguageTool checks only when user taps “Check”
- Works beautifully on mobile & web with responsive layout.

Tech & libraries:

- Expo SDK (latest), TypeScript, React Native, react-native-web
- Styling: no heavy UI kit; use StyleSheet + expo-linear-gradient + @expo/vector-icons
- Storage: @react-native-async-storage/async-storage
- Routing not required; single-screen is OK with bottom sheet/modal for Grammar
- Audio: use <Audio.Sound> from expo-av to play phonetic audio URLs
- Strict ESLint + Prettier config
- Design system (clean & modern)

Color palette:

- Primary: #5B8CFF (accents, pills, primary buttons)
- Surface: #0F1220 (dark-indigo background gradient at top) fading to #161A2B
- Card: #1E2340
- Text: #E8ECFF (high), #B9C1FF (muted)
- Positive chip / synonym: slightly lighter card with subtle border rgba(255,255,255,0.08)
- Components have 8/12/16px spacing grid, rounded corners (14-18), soft shadows (web: box-shadow; native: elevation)
- Buttons: pill-shaped, high touch targets (≥44px)

Typography:

- Title (word): 28-32 semibold
- Body: 16-18 regular
- Small meta (POS, phonetic): 13-14 medium, muted color

File structure (generate all files with content):

- /src
- /components
- SearchBar.tsx // input + sug dropdown (Datamuse), debounced

- DefinitionCard.tsx // headword, IPA, audio, meanings list
- SynonymChips.tsx // horizontal wrap chips; tap to search
- GrammarPanel.tsx // modal/bottom sheet; LanguageTool results with tap-to-replace
- EmptyState.tsx // friendly starter state
- ErrorState.tsx // network / not-found UI
- LoadingSpinner.tsx
- /utils
- api.ts // all fetch helpers for the 4 services
- debounce.ts // generic debounce
- storage.ts // AsyncStorage helpers
- text.ts // formatting helpers
- /styles
- theme.ts // colors, spacing, shadows, text styles
- App.tsx
- app.json
- package.json
- .eslintrc.cjs
- .prettierrc
- README.md

APIs (implement in utils/api.ts)

- getSuggestions(query: string): Promise<Suggestion[]>GET https://api.datamuse.com/sug?s=<query> → [{ word, score }]
- getSynonyms(word: string): Promise<string[]>GET https://api.datamuse.com/words?rel_syn=<word> → map to word
- getDefinitions(word: string): Promise<DictionaryResult>GET https://api.dictionaryapi.dev/api/v2/entries/en/<word>Parse: phonetics (text, audio), meanings (partOfSpeech, definitions[], examples)
- checkGrammar(text: string): Promise<LanguageToolResponse>POST https://api.languagetool.org/v2/checkBody: text, language=en-US (application/x-www-form-urlencoded)Return matches with offsets, replacements

- getWiktionaryExtract(word: string): Promise<string | null>GET https://en.wiktionary.org/w/api.php?action=query&prop=extracts&explaintext=1&format=json&origin=*&titles=<word>Return plain extract string or null

Caching

- In storage.ts implement:
- getCachedWord(word: string) / setCachedWord(word, payload)
- Cache shape: { word, fetchedAt, defs, syns, audio }
- TTL: 7 days; memory map for current session
- Only cache final “submitted” words (not every keystroke).

SearchBar.tsx (behavior)

- Controlled TextInput with debounce for suggestions
- Show dropdown panel with up to 6 suggestions (use keyboard and touch)
- Enter/submit selects the highlighted suggestion or current text
- On submit: call getDefinitions; meanwhile render LoadingSpinner
- Provide a small trailing button to clear and to open mic (optional: placeholder icon without impl)

DefinitionCard.tsx (behavior)

- Receives normalized data: { word, phonetics[], meanings[] }
- Phonetics line: show IPA (first available), play icon (if audio) → uses expo-av
- Meanings grouped by part of speech (limit 3 defs per POS; expand toggle)
- “Open in Thesaurus” button scrolls to SynonymChips
- If no data, render ErrorState with retry
- If 404, do Wiktionary extract fallback section (collapsed by default)

SynonymChips.tsx

- Load synonyms lazily (after definitions resolved). Show shimmer/loading
- Chips wrap; tapping a chip triggers a new lookup and scrolls to top

GrammarPanel.tsx

- Presented as a modal (or bottom sheet) triggered by a top-right toolbar button

- Multiline TextInput, “Check grammar” button
- List LanguageTool matches in order of offset with excerpt highlighting
- Each suggestion shows message + first replacement chip; tap to replace text
- “Apply all” button applies first suggested replacement per match
- Respect public rate limits (disable button while awaiting result)

Accessibility & responsiveness

- All touch targets ≥44px, labels for icons, test with keyboard on Web
- Layout scales to tablets (content max-width 840 on Web/tablet)

Networking & robustness

- Use AbortController to cancel in-flight suggestion queries on new keystrokes
- Guard against CORS and network errors with friendly ErrorState
- Timeouts: 8s for fetches; show retry
- If user enters spaces or punctuation, trim and lower-case word before lookup

Top-level screen layout (App.tsx)

- Top gradient header with app title LexiGo and toolbar icons:
- “Spell/Grammar” opens GrammarPanel
- “History” opens a simple recent search list (from cache) in a small modal
- Below: SearchBar, then either EmptyState, Loading, ErrorState, or Results
- Results shows DefinitionCard and SynonymChips

Styling polish

- Use expo-linear-gradient header (indigo → deep blue)
- Cards with subtle border rgba(255,255,255,0.06) and shadow
- Chips: pill, small shadow, hover style on web (increase elevation)

Audio

- If phonetics.audio exists, load with Audio.Sound.createAsync({ uri }), play on tap, and dispose on unmount.

Testing checklist (write in README)

- Typing quickly: suggestions update and no flicker
- Submit misspelling: suggestions + definition 404 → Wiktionary fallback appears
- Tap synonym: new word loads quickly, scroll to top
- Grammar: paste paragraph, get matches, tap to apply, verify text updates
- Offline/airplane mode: cached words still show; new lookups show error
- Web build: expo start --web works; responsive layout OK

Scripts

- dev: expo start
- web: expo start --web
- lint: ESLint
- Include standard Expo configs for web support

Deliverables

- All files above implemented with clean, commented code
- Attractive default favicon/app icon (simple letter “L” on indigo gradient)
- README with setup, usage, and API notes
- No API keys anywhere. All endpoints called client-side.

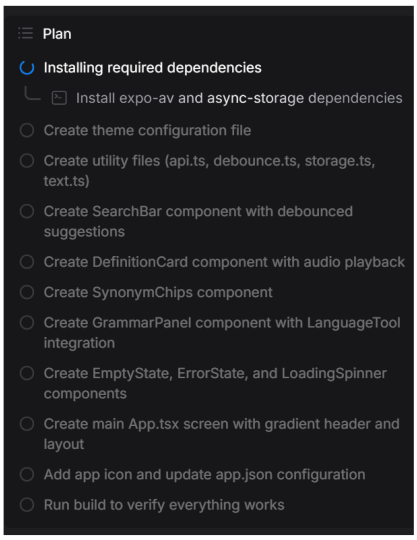
Acceptance criteria

- Works on iOS/Android/Web from Expo
- Suggest → Define → Synonyms → Grammar flow is smooth
- Audio playback for phonetics (when provided)
- Caching + debouncing implemented
- UI matches modern, minimal aesthetic described

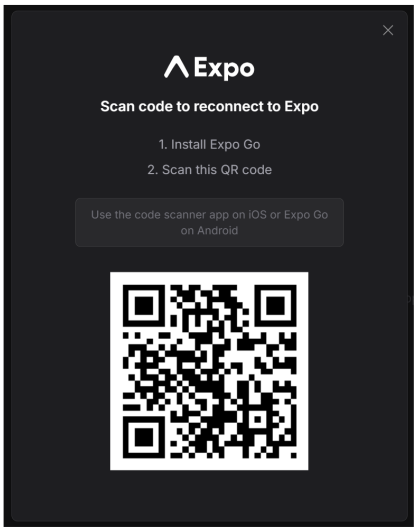
[End of Prompt]

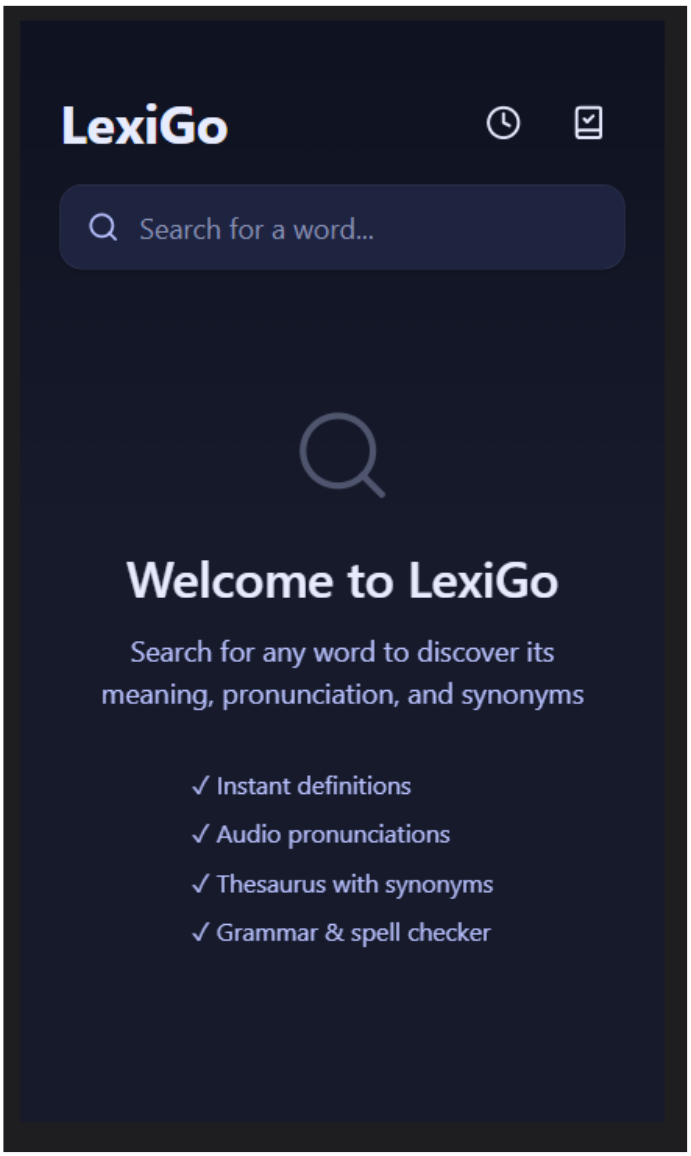
After pasting it into the Bolt.new chat, it started creating a comprehensive plan that addressed all of the details in my app plan, and here is the plan Bolt created before it began building the app.

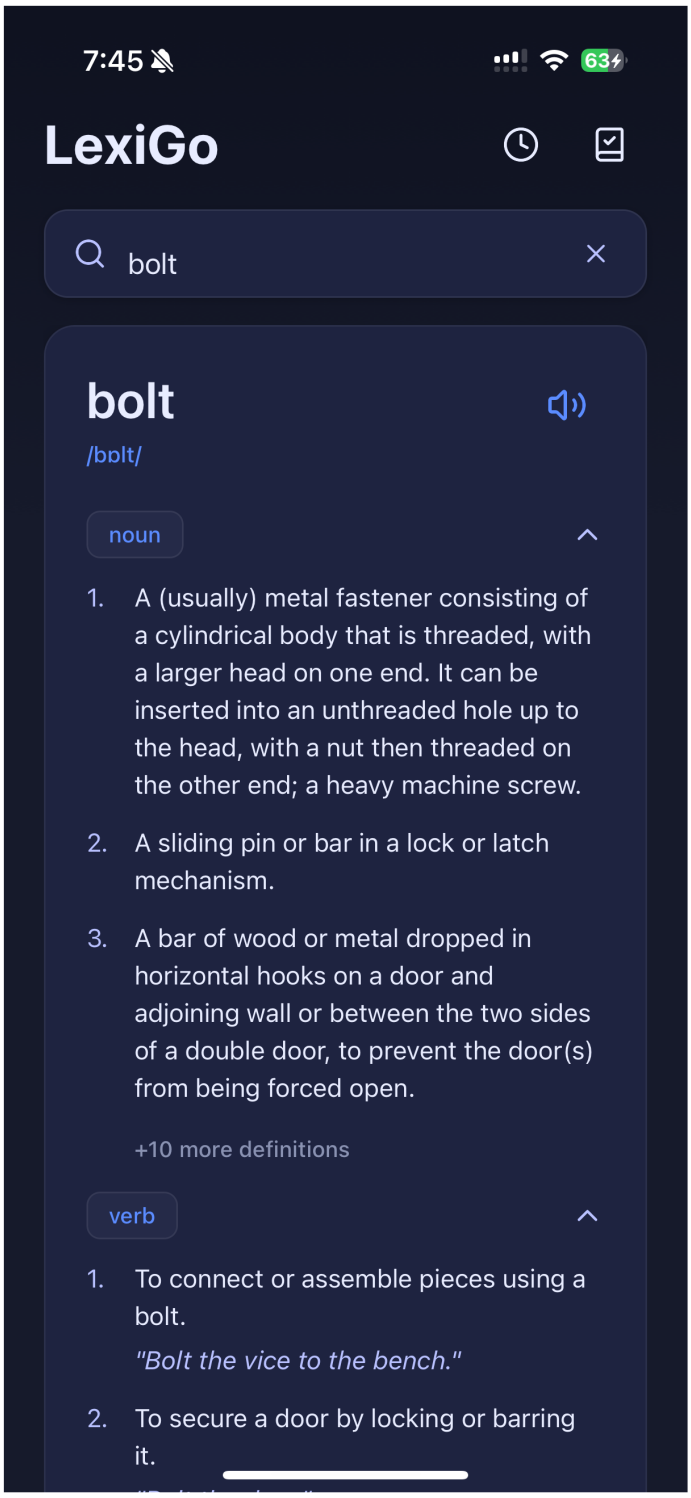
- Click the <> button to look through the code files, I can see my package-lock.json file has 10,000 lines of code in it.


Plan
Installing required dependencies
Install expo-av and async-storage dependencies
Create theme configuration file
Create utility files (api.ts, debounce.ts, storage.ts, text.ts)
Create SearchBar component with debounced suggestions
Create DefinitionCard component with audio playback
Create SynonymChips component
Create GrammarPanel component with LanguageTool integration
Create EmptyState, ErrorState, and LoadingSpinner components
Create main App.tsx screen with gradient header and layout
Add app icon and update app.json configuration
Run build to verify everything works

If you created an App using Expo which is compatible with both iOS and Android you will get a code to scan to view it on your device. Make sure to download the Expo Go app and scan the code.


Expo
Scan code to reconnect to Expo
1. Install Expo Go
2. Scan this QR code
Use the code scanner app on iOS or Expo Go on Android


LexiGo
Search for a word...
Welcome to LexiGo
Search for any word to discover its meaning, pronunciation, and synonyms
✓ Instant definitions
✓ Audio pronunciations
✓ Thesaurus with synonyms
✓ Grammar & spell checker


7:45
LexiGo
bolt
bolt
/bɒlt/
noun
1. A (usually) metal fastener consisting of a cylindrical body that is threaded, with a larger head on one end. It can be inserted into an unthreaded hole up to the head, with a nut then threaded on the other end; a heavy machine screw.
2. A sliding pin or bar in a lock or latch mechanism.
3. A bar of wood or metal dropped in horizontal hooks on a door and adjoining wall or between the two sides of a double door, to prevent the door(s) from being forced open.
+10 more definitions
verb
1. To connect or assemble pieces using a bolt.
"Bolt the vice to the bench."
2. To secure a door by locking or barring it.

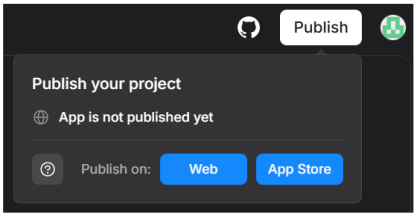
Inspect your results

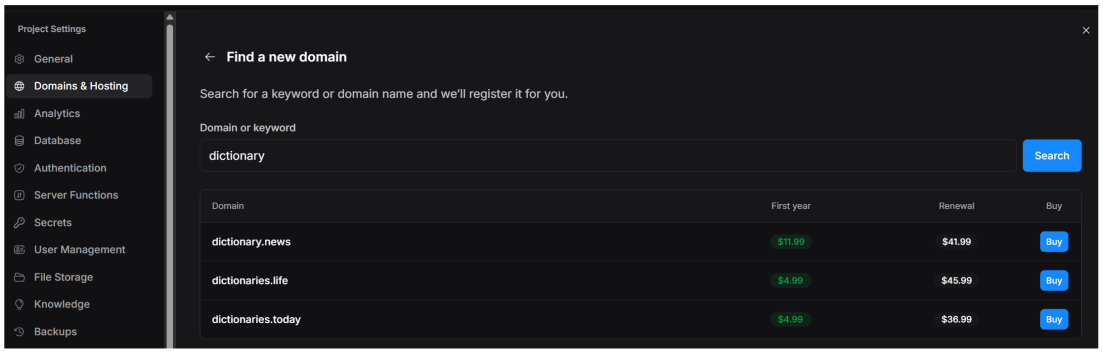
Here's the preview inside Bolt.new and on my phone using Expo Go. Some functionality is still not working such as speech, in order to save on credits I won't continue this project further but we can look at how to deploy it.

Share Your App

Click publish on the top right and choose web, this is the easiest way to allow others to begin using your app quickly, works the same as a mobile app but lives on a website. If you publish on Netlify you can buy a domain name directly from them and when you click deploy again inside of Bolt it will always update your endpoint website that you purchased.

If you are in testing or don't wish to buy a domain yet then rename your Netlify project to something more suitable, it will update your URL. If Netlify was not an option, you may also buy the domain from Bolt by choosing the custom domain under publish, buy a new domain.





Here's what it looks like when you buy directly from Bolt.

If you choose Publish Mobile App, Bolt will guide you through the publishing steps, but keep in mind that not every project can be released as a mobile or desktop app without extra work. Some apps need additional setup for device features, performance, permissions, or platform requirements before they are ready to ship. Bolt.new also lets you connect your project to GitHub so you can download the full codebase, store versions safely, and continue development in a different IDE or setup, such as Visual Studio Code on your own computer. If your project is built with web technologies, you can often package it as a desktop app using tools such as Electron or Tauri, which wrap your code into an installable app for Windows, macOS, and Linux.

You will also see other AI development platforms, such as Lovable, that focus on generating full projects and connecting them to common services like databases and payments. Before you commit to publishing, audit what your project is actually built with by asking the AI tool to explain the project's framework, file structure, and build output so you can confirm it matches your goals, and so you can use an internal or external AI chat to compare options, such as switching frameworks or packaging the app with Electron or Tauri. Because AI development tools change quickly, the best habit is to evaluate any platform by what it supports today, especially its export options, pricing, and how easily it connects to services such as Stripe, Supabase, and other APIs.

Bolt.new Project Ideas and Framework Pathways

Web UI apps

A single-page web app where most interaction happens in the browser after the first load. This is a strong fit for dashboards, tools, and interactive pages. Web UI apps often use a separate backend or third-party services for data.

React + Vite

A strong choice for building modern web user interfaces, especially when you want a project that feels like familiar apps students use. Example projects include a flashcards study app like Anki with decks, spaced repetition, and a daily streak, or a short-video feed prototype like TikTok with an infinite scroll feed, likes, comments mock, and a profile page. React is widely used for component-based interfaces, and it works smoothly with Vite for fast development, while React's component and state patterns fit feed-style layouts and dynamic lists. Websites using react include: Instagram, Facebook, and Discord. [1]

Vue + Vite

Works well for web apps that involve forms, planning, and data-driven screens. Example projects include a homework planner with a calendar view, priorities, and a printable weekly plan, or a recipe organizer with tags, favorites, and an auto-generated shopping list. Vue's reactive data model supports apps where the interface updates automatically when data changes, and it is a natural fit for filtering, sorting, and showing derived information. Websites using react include: GitLab, TikTok and Upwork. [2]

Svelte + Vite

A good option for lightweight interactive web apps that need responsive updates and clear state-driven behavior. Example projects include a Pomodoro timer with presets, session history, and a focus mode screen, or an interactive story app with branching choices and saved progress. Svelte compiles to efficient JavaScript, which can perform well for apps with frequent UI updates and multi-step flows. Websites using react include: Gamma, Note, and Netlify. [3]

Angular

Often chosen for larger, structured web applications that resemble professional internal tools. Example projects include a helpdesk ticket system interface with search, status filters, and a detail view, or an inventory tracker with check-in and check-out, low-stock alerts, and CSV export. Angular is a full, opinionated framework with built-in patterns for routing and forms, and it fits apps that need consistent architecture, validation, and complex tables. Websites using react include: Progressive, Amtrak, and Crunchbase. [4]

Full-stack web apps

A web app that includes both the user interface and server-side features such as routing, form handling, authentication flows, and database access, often inside one framework.

Next.js

A full-stack React framework that is common in production web apps and works well for projects that combine public pages with logged-in features. Example projects include a community forum like Reddit with posts, comments, upvotes, and user profiles, or a simple shop with products, cart, login, and a checkout placeholder. Next.js supports file-based routing and server features, which are useful for building real app flows and preparing for API and commerce integrations. Websites using react include: Nike, The Washington Post, and IGN. [5]

Remix

A full-stack React framework that is especially strong for form-heavy applications and data-driven workflows. Example projects include a student submission portal with login, assignment uploads or links, and teacher feedback, or an event registration system with multi-step forms and confirmation pages. Remix is built around web request and response patterns, which maps well to real registration, submission, and admin workflows. Websites using react include: Bolt.new, PayPal, and Open Ai. [6]

Nuxt

A full-stack Vue framework that is a strong choice when you want Vue plus server-side capabilities for dynamic content and user accounts. Example projects include a membership content site with login and gated lessons, or an event platform with event pages, RSVP, and an admin posting tool. Nuxt supports routing and server features that fit authenticated content, publishing tools, and database-backed pages. Websites using react include: Louis Vuitton, Virgin Galactic, and n8n. [7]

Astro

Optimized for content-first websites and performance, while still allowing interactive components where they make sense. Example projects include a textbook companion site with chapter pages, downloads, a glossary, and search, or a portfolio with case studies and a contact form. Astro is a strong choice when most of your pages are content, but you still want modern UI components in specific sections. Websites using react include: IKEA, Porsche, and Cloudfare. [8]

React Native with Expo

A practical way to build real iOS and Android apps from one codebase using React Native, with a workflow that simplifies device testing and builds. Example projects include vocabulary flashcards with swipe study mode, daily practice, and offline decks, or a habit tracker with streaks and reminders placeholder. Expo helps you move from a working prototype to a mobile app build, and it supports adding device capabilities through Expo's APIs as your project grows. Websites using react include: ChatGPT, Shopify, and Stripe. [9]

Additional Solutions

Outside of Bolt.new, you can export your project to GitHub and download the full codebase, then use packaging tools to reach other platforms. For desktop apps, Electron or Tauri can package a web app into an installable app for Windows, macOS, and Linux. For mobile apps, Capacitor or Cordova can wrap a web app into an iOS and Android app shell using a WebView plus plugins.

Chapter 3 Make an App Using Bolt.new Rubric

This chapter focused on using Bolt.new to generate an app project, review the created files, test the app, revise problems, and document the development process.

Category	Points	Evidence of Completion
Project Setup	5	A Bolt.new project is created, clearly named, saved, and prepared using an appropriate app framework such as React Native with Expo.
Prompt and App Planning	5	A clear app prompt is written with the project goal, required features, design preferences, and expected user interactions.
App Build	5	The app is generated with a usable structure, including organized files, visible interface elements, and the required main features.
Testing and Revision	5	The app is tested in the preview area or on a device. Errors, missing features, layout problems, or confusing behavior are identified and revised.
Evidence of Work	5	Screenshots, project links, prompt examples, file screenshots, or other project proof are submitted.
Reflection	5	A short explanation describes the app idea, how the prompt affected the result, what was tested or revised, and relevant computer science vocabulary such as framework, component, file structure, user interface, API, package, debugging, or iteration.
Total	**30**	

Submission Checklist

Requirement	Check
Bolt.new project created	☐
Project clearly named and saved	☐
App prompt included	☐
App interface generated	☐
Required app features included	☐
Project files reviewed	☐
App tested in preview or on a device	☐
At least one error, missing feature, or design issue revised	☐
Screenshots or project proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the app created in Bolt.new. Explain how the prompt helped shape the app features, design, or file structure. Identify one part of the app that was tested, fixed, or improved before final submission.*

1. Wappalyzer, “React: Websites Using React,” Wappalyzer, accessed February 20, 2026, https://www.wappalyzer.com/technologies/javascript-frameworks/react/

2. Wappalyzer, “Vue.js: Websites Using Vue.js,” Wappalyzer, accessed February 20, 2026, https://www.wappalyzer.com/technologies/javascript-frameworks/vue-js/

3. Wappalyzer, “Svelte: Websites Using Svelte,” Wappalyzer, accessed February 20, 2026, https://www.wappalyzer.com/technologies/javascript-frameworks/svelte/

4. Made with Angular, “A Showcase of Web Apps Made with Angular,” Made with Angular, accessed February 20, 2026, https://www.madewithangular.com/

5. Vercel, “Next.js Showcase,” Next.js, accessed February 20, 2026, https://nextjs.org/showcase

6. Wappalyzer, “Remix: Websites Using Remix,” Wappalyzer, accessed February 20, 2026, https://www.wappalyzer.com/technologies/web-frameworks/remix/

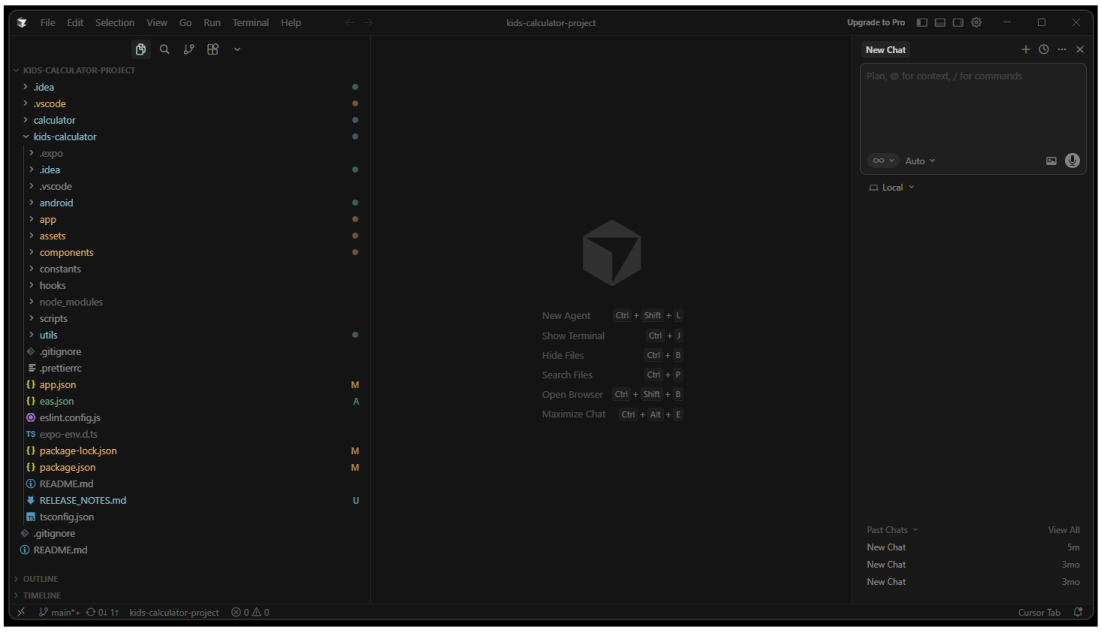
7. Nuxt, “Nuxt Showcase: Real-World Web Applications Built with Nuxt,” Nuxt, accessed February 20, 2026, https://nuxt.com/showcase

8. Astro, “Astro Showcase,” Astro, accessed February 20, 2026, https://astro.build/showcase/

9. React Native, “React Native Showcase: Who Is Using React Native?,” React Native, accessed February 20, 2026, https://reactnative.dev/showcase

4

Build a React Expo App with Cursor



In this lesson you will move from browser based app builders to a real local development workflow using Cursor, an AI powered code editor that works like Visual Studio Code. You will create an Expo React Native project on your own computer, then use Cursor's AI to generate a simple calculator interface and logic. Next you will run the Expo development server and test the app on the web first so you can quickly confirm that the layout and buttons work as expected. After that you will open the same project on a phone using Expo Go and a QR code so you can see how your app behaves on a real device. This lesson is designed to help you understand how real app projects are structured, where files are saved, and how developers use terminals and commands to build and test software across multiple platforms.

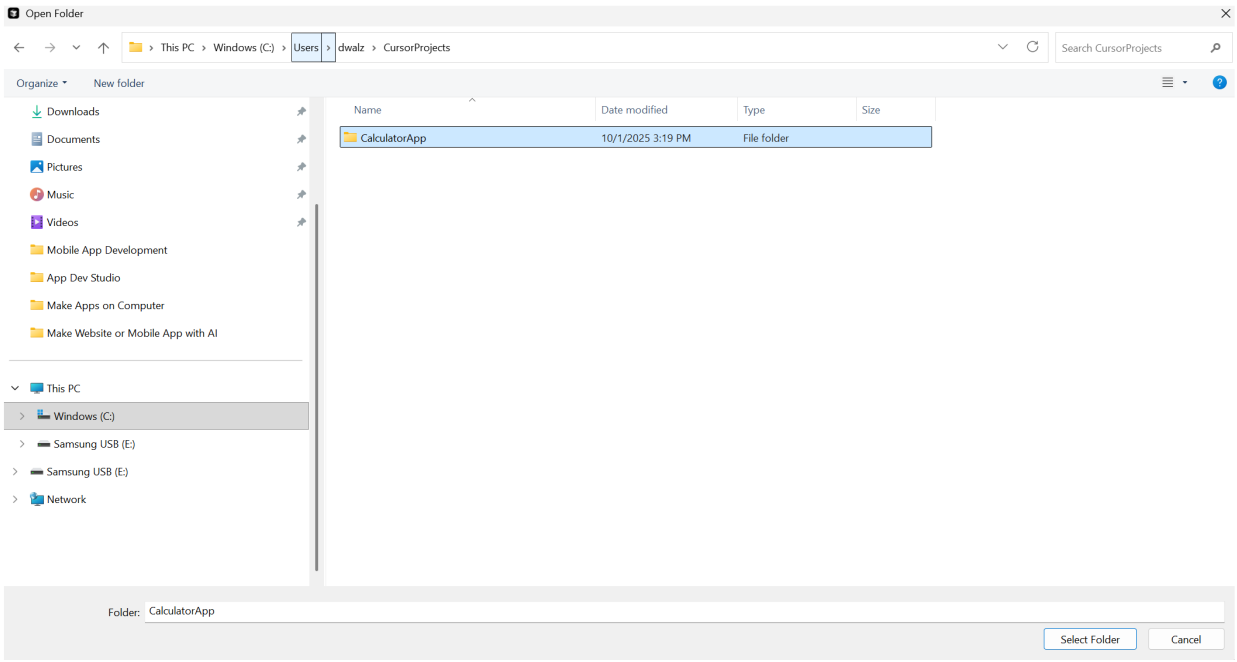
Install Cursor

- Open this link and download Cursor: https://cursor.com
- Run the installer, then open Cursor.

Create Your Project Folder

Create a folder in your computer called CursorProjects, then inside that folder create another folder and name it the name of your project. Here's what my folder path looks like.

C:\Users\dwalz\CursorProjects



Press select folder.

Now your Cursor project will save any additional files or code we create inside that folder.

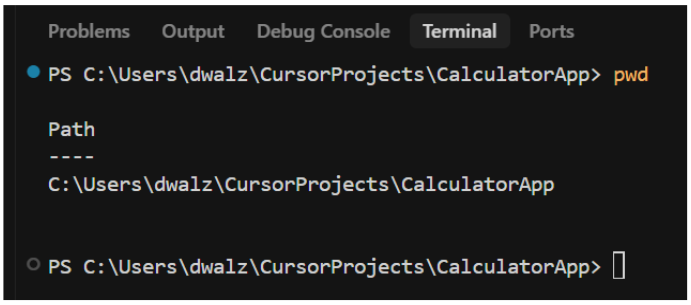
Install Node.js (LTS): lets your computer run JavaScript tools. Download: https://nodejs.org

Open the Integrated Terminal in Cursor

- Open the terminal:
- At the top of the screen click Terminal and then New Terminal
- Confirm you're inside the CalculatorApp folder by printing the current path:

Inside the terminal type:

pwd



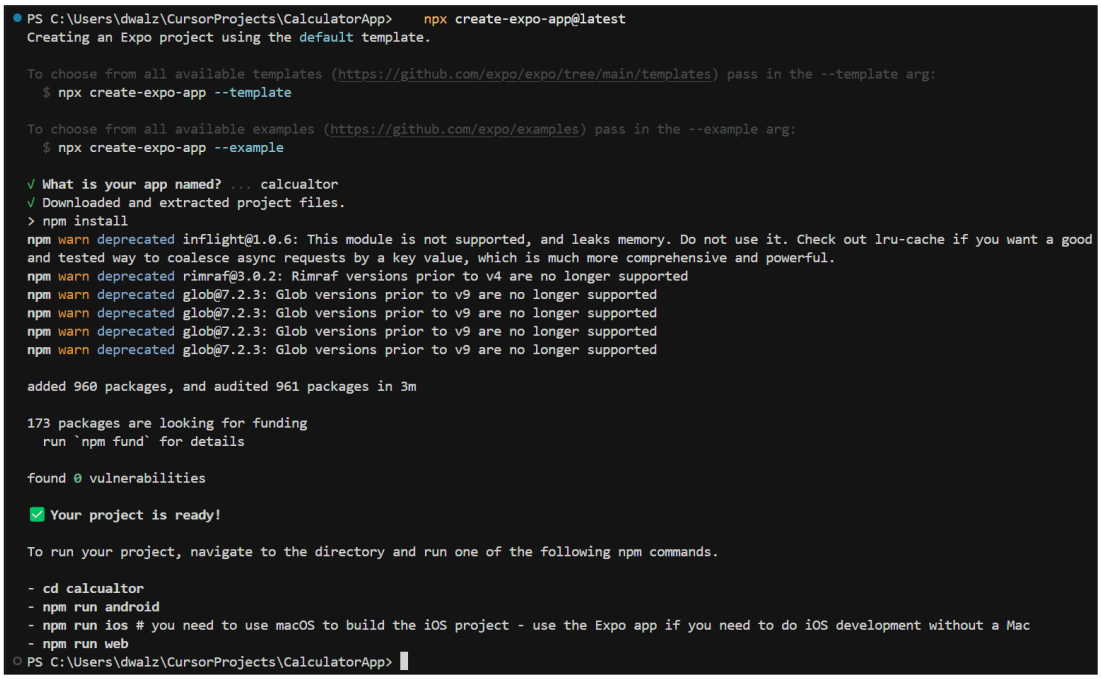
This confirms where the project files are

Create Expo Project

Run this command in the terminal:

```
npx create-expo-app@latest
```

Do not choose a template, just enter the name of the app which is: calculator


PS C:\Users\dwalz\CursorProjects\CalculatorApp> npx create-expo-app@latest
Creating an Expo project using the default template.
To choose from all available templates (https://github.com/expo/expo/tree/main/templates) pass in the --template arg:
$ npx create-expo-app --template
To choose from all available examples (https://github.com/expo/examples) pass in the --example arg:
$ npx create-expo-app --example
√ What is your app named? ... calcualtor
√ Downloaded and extracted project files.
> npm install
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
added 960 packages, and audited 961 packages in 3m
173 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
Your project is ready!
To run your project, navigate to the directory and run one of the following npm commands.
- cd calcualtor
- npm run android
- npm run ios # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development without a Mac
- npm run web
PS C:\Users\dwalz\CursorProjects\CalculatorApp>

It may take a few minutes to complete the installation, if there's an error code copy and paste it into an AI chat to resolve it.

Sometimes Windows blocks npm with a red error message like: npm.ps1 cannot be loaded because running scripts is disabled on this system. If you see this error, follow the steps below. If not, skip ahead to the next step, Use Cursor's AI to Generate the Calculator.

Open PowerShell

Press the Windows key on your keyboard.

Type PowerShell.

Right-click Windows PowerShell → choose Run as Administrator.

Type this command and press Enter:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

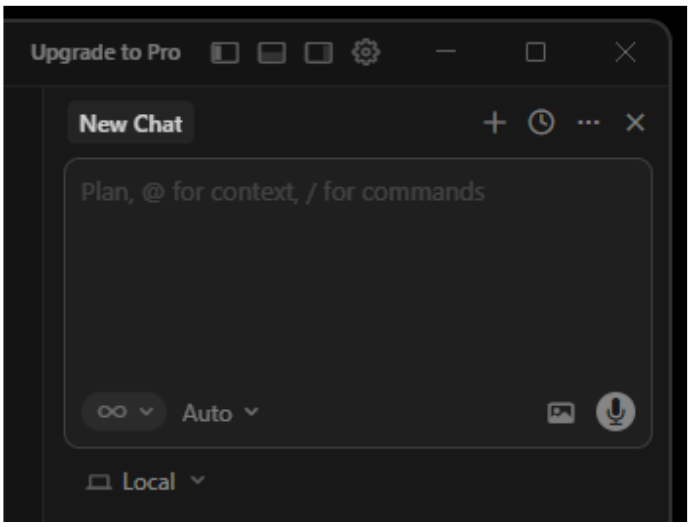
Press Y (Yes) if asked to confirm.

Close PowerShell.

Reopen Cursor and try the Expo command again.

Use Cursor's AI to Generate the Calculator

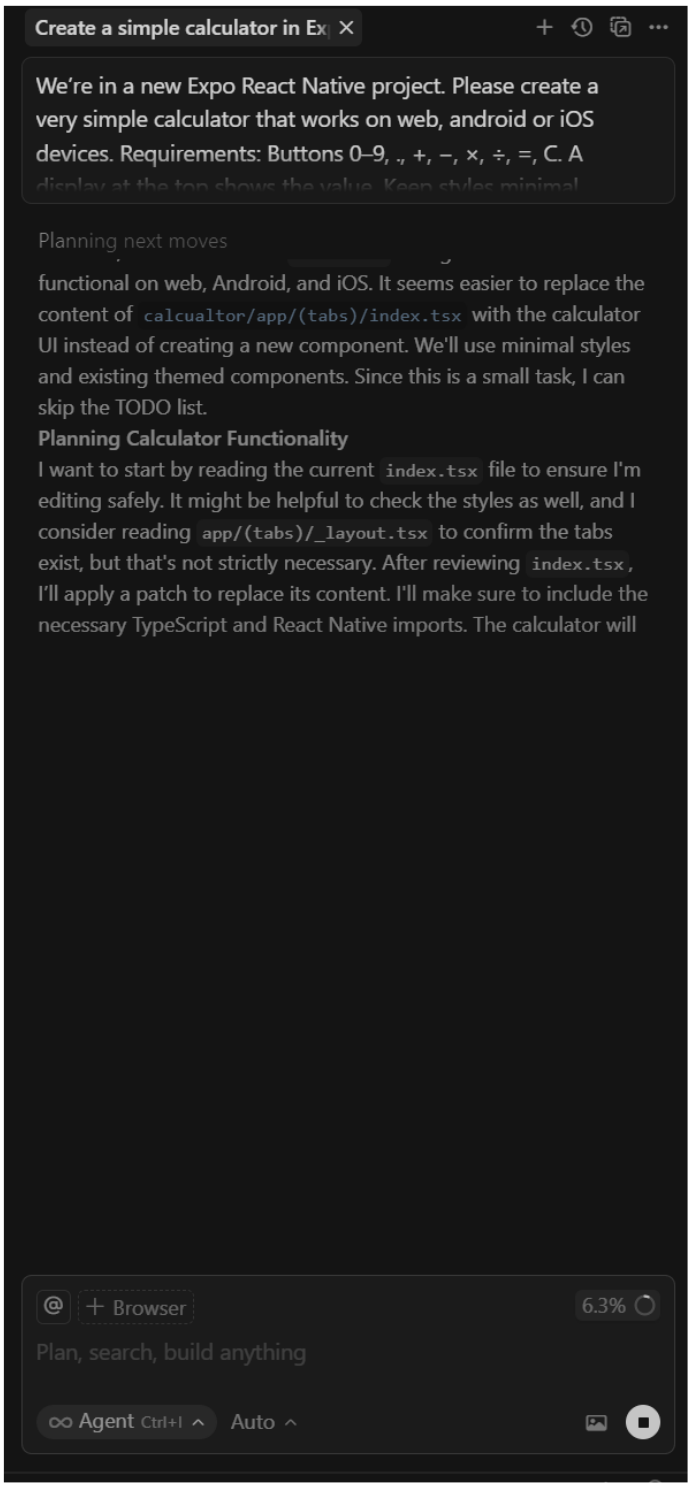
The AI panel may be open be default inside cursor, if you do not see it type: Ctrl + L



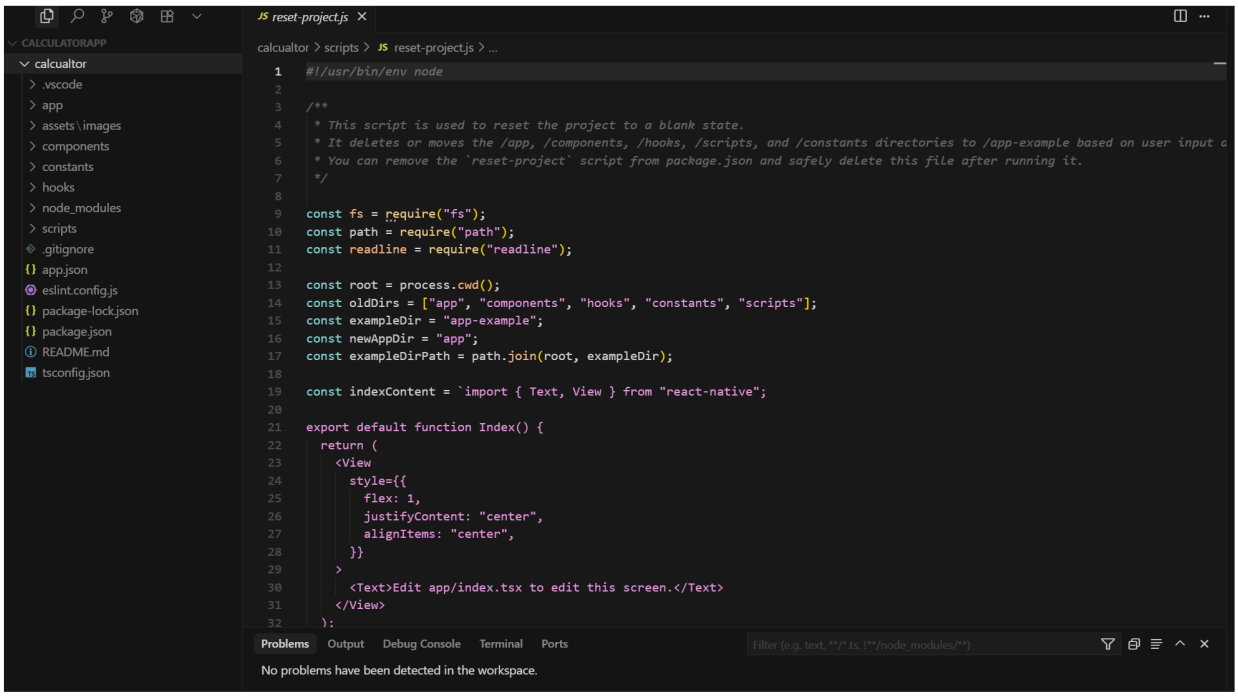
Paste this prompt into the AI chat or try your own:

```
We're in a new Expo React Native project. Please create a very simple calculator that works on web, android or iOS devices. Requirements: Buttons 0-9, ., +, −, ×, ÷, =, C. A display at the top shows the value. Keep styles minimal.
```

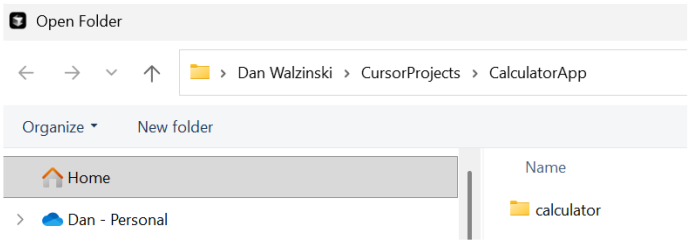
It will start working and look like this


Create a simple calculator in Ex
We're in a new Expo React Native project. Please create a very simple calculator that works on web, android or iOS devices. Requirements: Buttons 0–9, ., +, −, ×, ÷, =, C. A
Planning next moves
functional on web, Android, and iOS. It seems easier to replace the content of calcualtor/app/(tabs)/index.tsx with the calculator UI instead of creating a new component. We'll use minimal styles and existing themed components. Since this is a small task, I can skip the TODO list.
Planning Calculator Functionality
I want to start by reading the current index.tsx file to ensure I'm editing safely. It might be helpful to check the styles as well, and I consider reading app/(tabs)/_layout.tsx to confirm the tabs exist, but that's not strictly necessary. After reviewing index.tsx, I'll apply a patch to replace its content. I'll make sure to include the necessary TypeScript and React Native imports. The calculator will
@
+ Browser
6.3%
Plan, search, build anything
Agent Ctrl+I
Auto

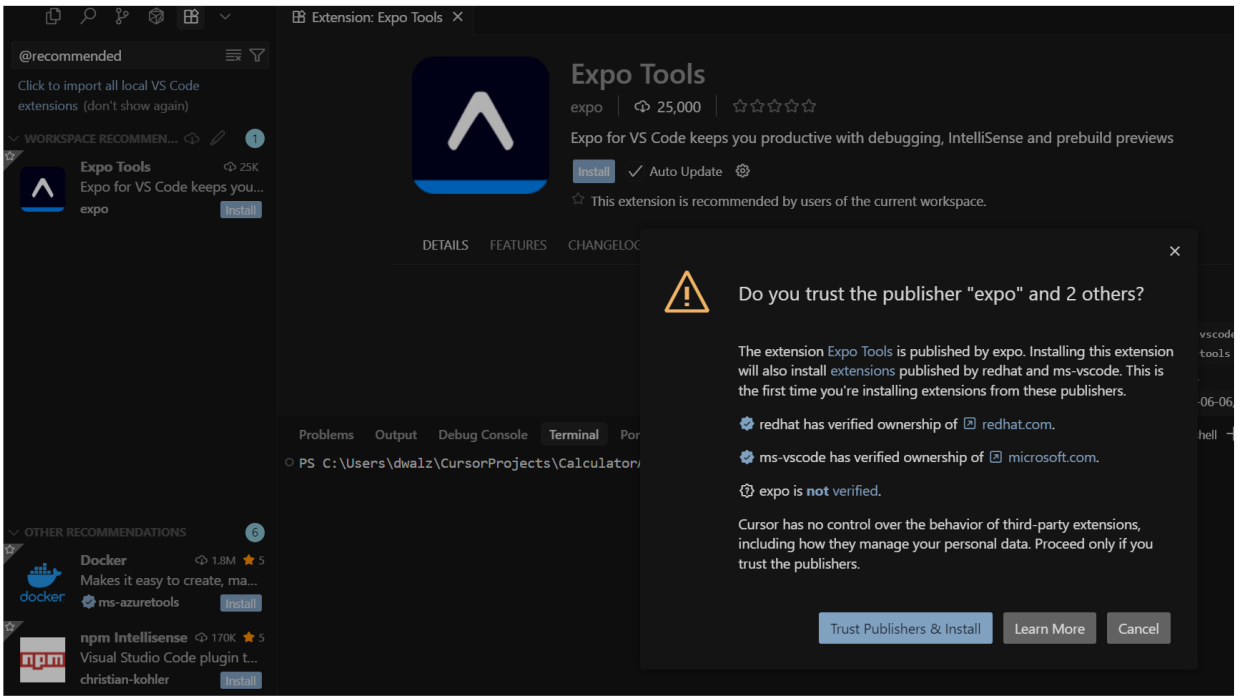
- When the AI shows file changes, click Apply (or Create/Replace file) so App.js is saved into your folder.
- View your project files in the left pane. Click through the different files to view them.



Likely cursor created one master folder for your project (above you can see it as calculator) but cursor is currently opened under the folder we created earlier so you need to go to file, open folder, and select the new master folder created by Cursor, plus I renamed it because the spelling was wrong.



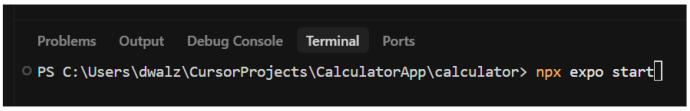
Because the code contains needed Expo Tools you can install them if prompted to do so.


Extension: Expo Tools
@recommended
Click to import all local VS Code extensions (don't show again)
WORKSPACE RECOMMEN...
Expo Tools
Expo for VS Code keeps you...
expo
Install
OTHER RECOMMENDATIONS
Docker
Makes it easy to create, ma...
ms-azuretools
npm Intellisense
Visual Studio Code plugin t...
christian-kohler
Expo Tools
expo
25,000
Expo for VS Code keeps you productive with debugging, IntelliSense and prebuild previews
Install
Auto Update
This extension is recommended by users of the current workspace.
DETAILS
FEATURES
Problems
Output
Debug Console
Terminal
PS C:\Users\dwalz\CursorProjects\Calculator
Do you trust the publisher "expo" and 2 others?
The extension Expo Tools is published by expo. Installing this extension will also install extensions published by redhat and ms-vscode. This is the first time you're installing extensions from these publishers.
redhat has verified ownership of redhat.com.
ms-vscode has verified ownership of microsoft.com.
expo is not verified.
Cursor has no control over the behavior of third-party extensions, including how they manage your personal data. Proceed only if you trust the publishers.
Trust Publishers & Install
Learn More
Cancel

Run the App on the Web

Start the Expo dev server, type this inside the terminal in cursor:

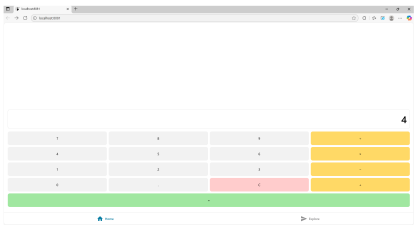
```
npx expo start
```



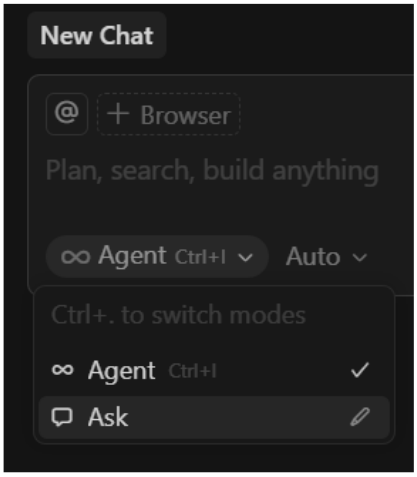
Your terminal will look like this, you can drag the panel edge up to see it in a larger view.

Problems
Output
Debug Console
Terminal
Ports
PS C:\Users\dwalz\CursorProjects\CalculatorApp\calculator> npx expo start
Starting project at C:\Users\dwalz\CursorProjects\CalculatorApp\calculator
React Compiler enabled
Starting Metro Bundler
› Metro waiting on exp://192.168.0.107:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
› Web is waiting on http://localhost:8081
› Using Expo Go
› Press s | switch to development build
› Press a | open Android
› Press w | open web
› Press j | open debugger
› Press r | reload app
› Press m | toggle menu
› shift+m | more tools
› Press o | open project code in your editor
› Press ? | show all commands
Logs for your project will appear below. Press Ctrl+C to exit.

You can simple press the buttons listed in the menu to run those commands but first you should test open web (press w) to see if the app looks like how you expected it to.



Here's what mine looks like, if it's not working for you copy and paste any information either in the terminal or on the site which you can ask AI to resolve, this can be done with an external AI or by using the Ask Agent option in the right AI panel in Cursor.



If you need a clean reload, stop the server (Ctrl+C) and start with cache cleared type:

```
npx expo start -c
```

Note: "Networking has been disabled" warning only affects phone testing — web still works.

If Errors Copy & Paste Error Messages

Terminal copy/paste shortcuts

- `Windows: Copy = Ctrl+Shift+C, Paste = Ctrl+Shift+V`
- `Mac: Copy = Cmd+C, Paste = Cmd+V`

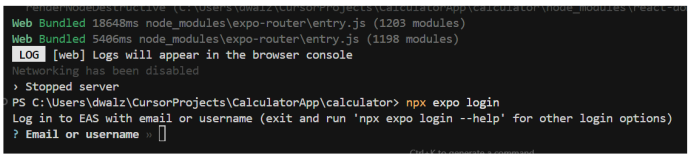
View App on your Phone using Expo Go

Create and log into an Expo account

- Sign up: https://expo.dev
- After creating an account inside Cursor in the terminal press (control + c) to stop running the app and to log in type:

```
npx expo login
```

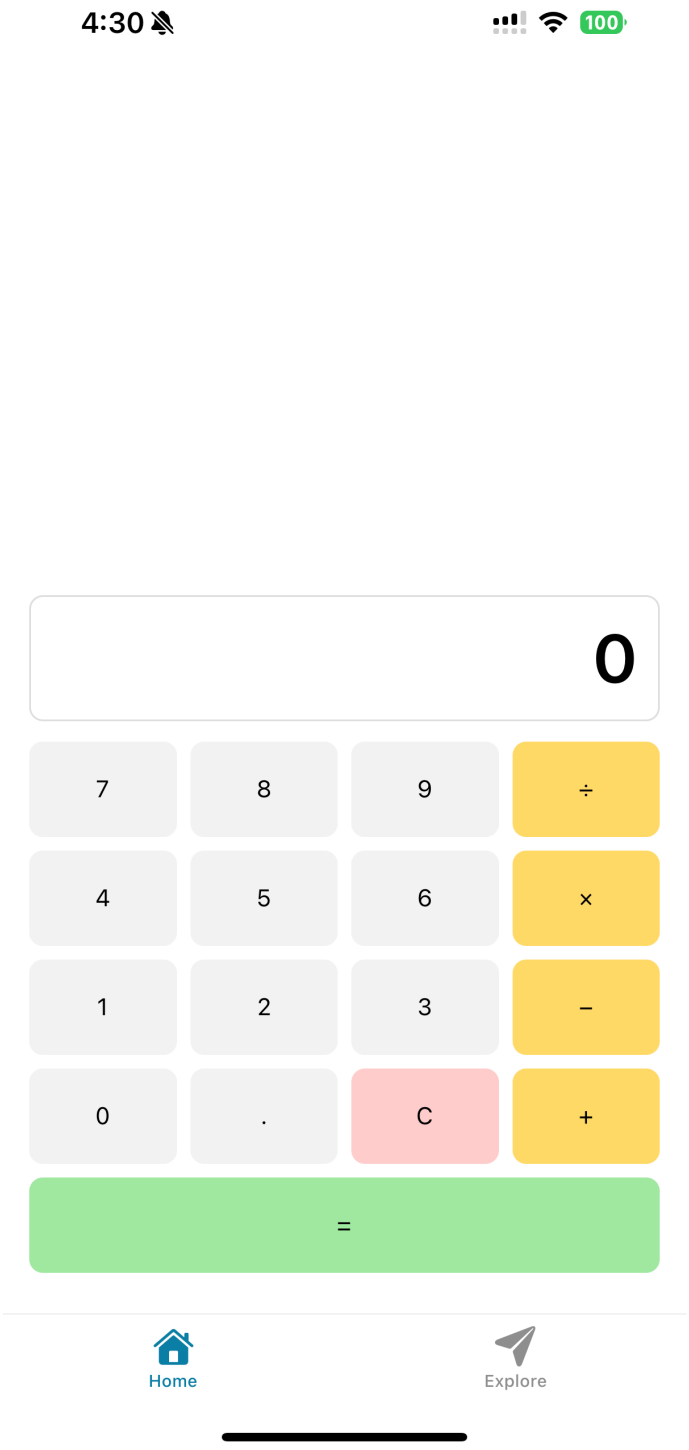
- Enter your Expo username and password


Web Bundled 18648ms node_modules\expo-router\entry.js (1203 modules)
Web Bundled 5406ms node_modules\expo-router\entry.js (1198 modules)
LOG [web] Logs will appear in the browser console
Networking has been disabled
› Stopped server
PS C:\Users\dwalz\CursorProjects\CalculatorApp\calculator> npx expo login
Log in to EAS with email or username (exit and run 'npx expo login --help' for other login options)
? Email or username

Type in terminal:

```
npx expo start
```

- Install Expo Go from the app store, login to the account you just created
- Use your camera to scan the QR code in the Cursor terminal, it should open Expo Go on your phone and the app you just created. Below is a screenshot from my phone.



Test your calculator on the web and on your phone to confirm that every button works as expected. If you see an error, copy the full message from the terminal or screen and paste it into your editor or AI chat so you can troubleshoot it step by step. When you are satisfied with the results, share a screenshot of your app with your teacher or friends. If you want to improve the design or add new features, continue iterating on the project and test again after each change. The table below summarizes what you can build with a code editor like Cursor and which toolchains are required to run each type of project. Choose one category that interests you, then research what tools you would need and what steps it would take to create that kind of app.

Cursor as a Code Editor: What you can build with it			
Category	**Builds**	**Typical Stack**	**Toolchain**
Web	Modern web UI apps	React, Vue, or Svelte with Vite	Node.js LTS + npm/pnpm + framework CLI.
Web	Full stack web apps	Next.js, Remix, Nuxt, Astro	Node.js LTS + full-stack framework tooling.
Mobile	Cross platform mobile apps	React Native with Expo	Expo CLI/dev server + Expo Go
Mobile	Cross platform mobile apps	Flutter (Dart)	Flutter SDK + Android SDK; iOS: macOS + Xcode.
Mobile	Native iOS apps	Swift and SwiftUI	Xcode + Apple SDKs
Mobile	Native Android apps	Kotlin and Jetpack Compose	Android SDK + Gradle
Desktop	Windows desktop apps	C# and .NET with WinUI 3, WPF, or WinForms	.NET SDK + Visual Studio + Windows App SDK.
Desktop	Cross platform desktop apps	Electron with JavaScript or TypeScript	Node.js + Electron.
Desktop	Lightweight cross platform desktop apps	Tauri with a web UI plus a Rust backend	Rust + platform deps; Windows: C++ tools + WebView2.
Mobile + Desktop	Cross platform native apps	.NET MAUI	.NET SDK + .NET MAUI + platform SDKs per target.
Games	2D and 3D games	Unity	Unity Hub + Unity Editor.
Games	High end 3D games	Unreal Engine	Epic Games Launcher + Unreal Engine.
Games	2D and 3D games, open source	Godot	Godot Editor

Chapter 4 Build a React Expo App with Cursor Rubric

This chapter focused on creating a React Expo app with Cursor, using the terminal to run the project, testing the app in a browser or on a device, revising problems, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	A React Expo project is created in Cursor. The project folder is clearly named, opened correctly, and prepared with the required files and dependencies.
Terminal and Development Workflow	5	The correct terminal commands are used to create, install, start, and run the Expo project. The app opens successfully in the browser, simulator, emulator, or Expo Go.
App Build	5	The app includes the required interface and features, such as a working calculator layout, display area, buttons, and organized React Native code.
Testing and Revision	5	The app is tested after running. Errors, layout issues, missing features, or broken interactions are identified and revised.
Evidence of Work	5	Screenshots, project files, terminal output, browser preview, device preview, or other project proof are submitted.
Reflection	5	A short explanation describes the app, the development workflow, the testing process, and relevant computer science vocabulary such as terminal, command, dependency, component, state, event, debugging, Expo, or React Native.
Total	**30**	

Submission Checklist

Requirement	Check
Cursor installed and opened	☐
React Expo project created	☐
Project folder clearly named	☐
Required terminal commands completed	☐
App opened in browser, simulator, emulator, or Expo Go	☐
App interface and required features completed	☐
App tested after running	☐
At least one error, layout issue, or feature issue revised	☐
Screenshots or project proof included	☐
Short reflection completed	☐

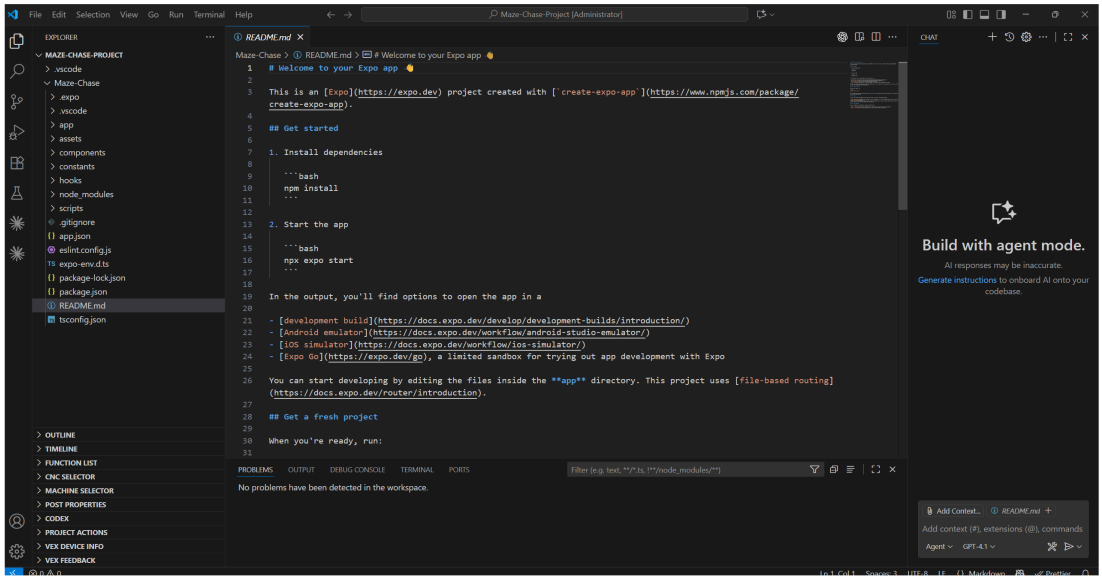
Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the React Expo app created with Cursor. Explain how the terminal was used during the development process. Identify one error, design issue, or feature problem that was tested and improved before final submission.*

5

Make Your Own Windows App in VS Code



In this lesson you will use Visual Studio Code to create a real Windows desktop app using Python. You will install Python and VS Code, set up a simple project folder, and write a small calculator program with a basic graphical interface. After you run and test the app from inside VS Code, you will learn how to package it into a standalone Windows application file so it can run like a normal program without opening your editor. By the end of the lesson you will understand the difference between writing code in a code editor and using the correct toolchain to run and package your project for Windows.

Install Python

- Go to the official site: https://www.python.org/downloads/
- Download the latest version of Python.
- Run the installer. Important: check the box that says 'Add python.exe to PATH' before clicking Install Now.

Install Python 3.13.7 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
Install Now
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
Use admin privileges when installing py.exe
Add python.exe to PATH
Cancel

To test if it worked:

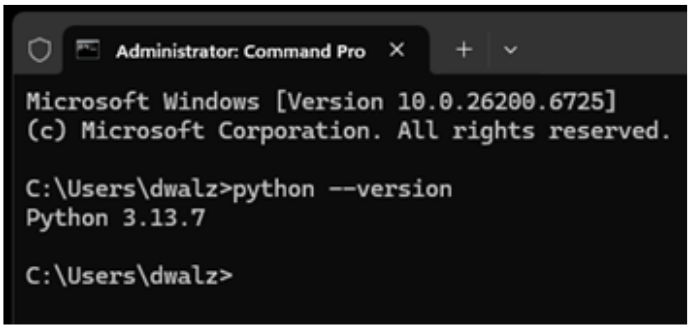
• Press Windows Key on your keyboard (bottom left).

• Type 'cmd' and hit Enter. This opens the Command Prompt (a black box where you type commands).

• Type:

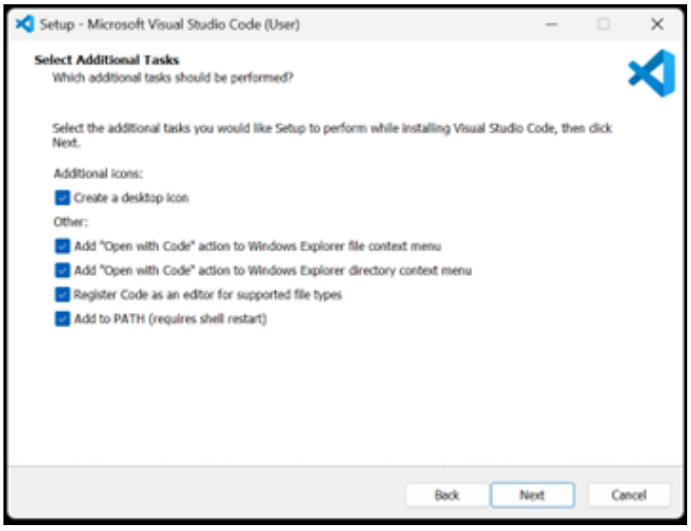
```
python --version
```

You should see something like Python 3.12.3.



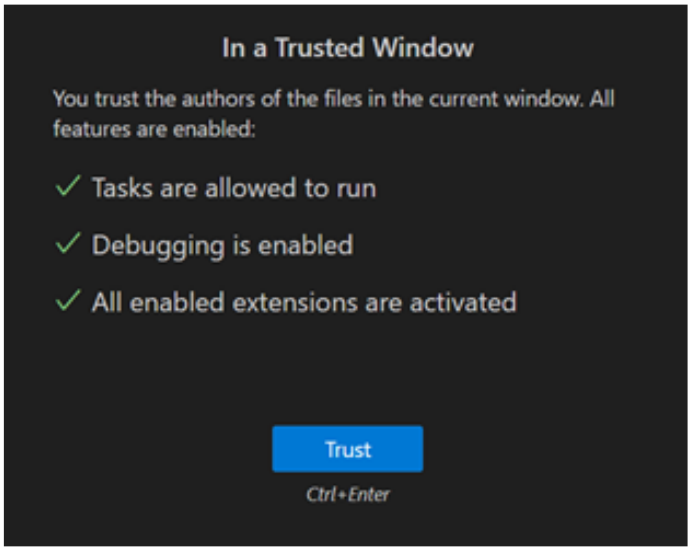
Install Visual Studio Code

- Download here: https://code.visualstudio.com/
- Open the installer → enable all options → Install.


Setup - Microsoft Visual Studio Code (User)
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.
Additional icons:
Create a desktop icon
Other:
Add "Open with Code" action to Windows Explorer file context menu
Add "Open with Code" action to Windows Explorer directory context menu
Register Code as an editor for supported file types
Add to PATH (requires shell restart)
Back
Next
Cancel

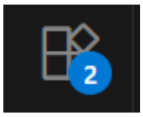
After installing, open VS Code, if you see this at the top click manage.

If you see this at the top click manage and enable trust.

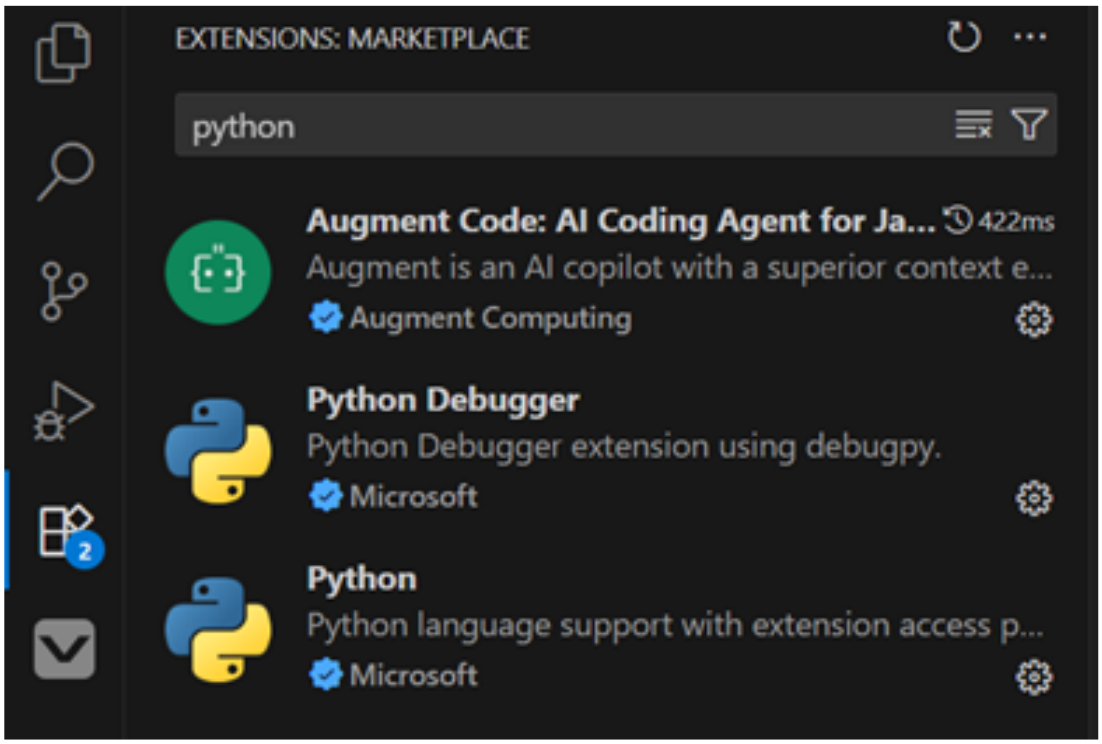

In a Trusted Window
You trust the authors of the files in the current window. All features are enabled:
Tasks are allowed to run
Debugging is enabled
All enabled extensions are activated
Trust
Ctrl+Enter

Sign in or create an account, I connected mine to Github. I saw an option to use a tool called Augment but I decided not to use it because it costs at least $20 a month.

On the left sidebar, click the Extensions icon, it looks like four squares.



Search for Python by Microsoft and click Install.



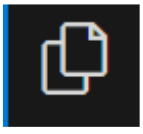
Create a new project

On your Desktop, right-click → New → Folder → name it Python-Calculator.

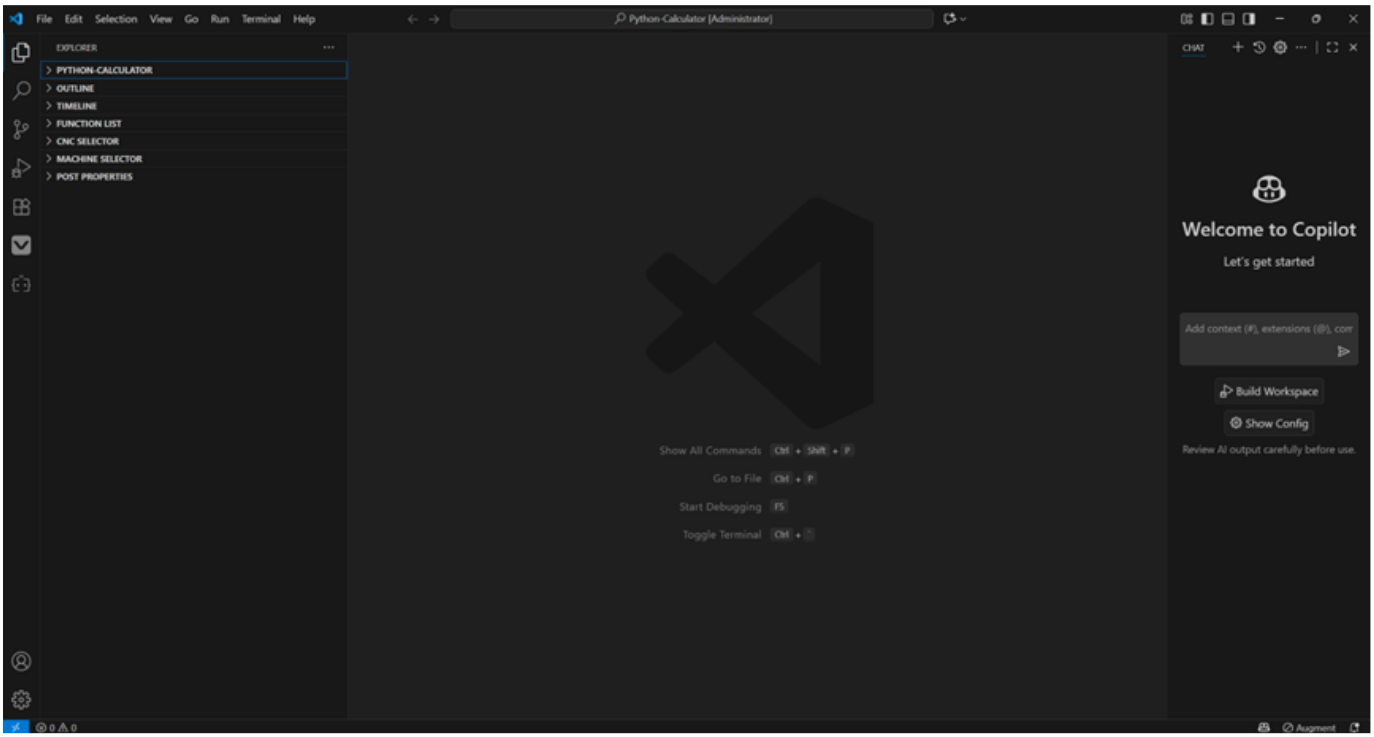
Note: This is great for quick projects, for longer projects create a file to store them in your C: drive or flash drive. Ex: C:\Users\dwalz\VS Code Projects\Python-Calculator.

Open VS Code → go to File → Open Folder… → pick Python-Calculator.

Click the Explorer tab on the top left.

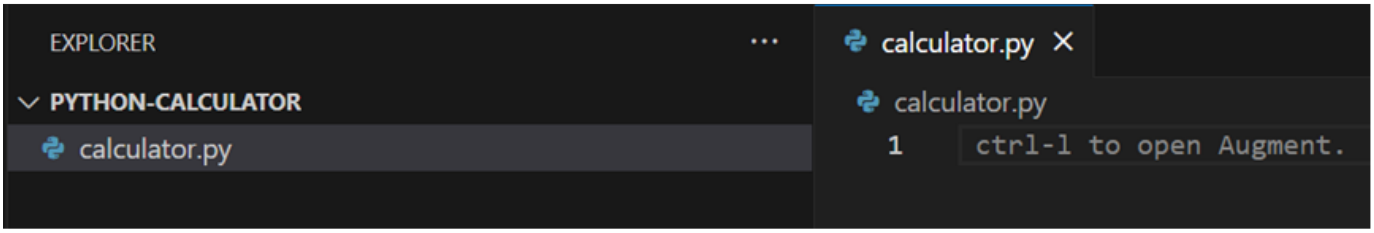


Clean up your workspace by closing any windows you don't need, here's what mine looks like now, you can see my file: Python-Calculator is open in the explorer, any files or code we create will be added there.



Create Your Python Files

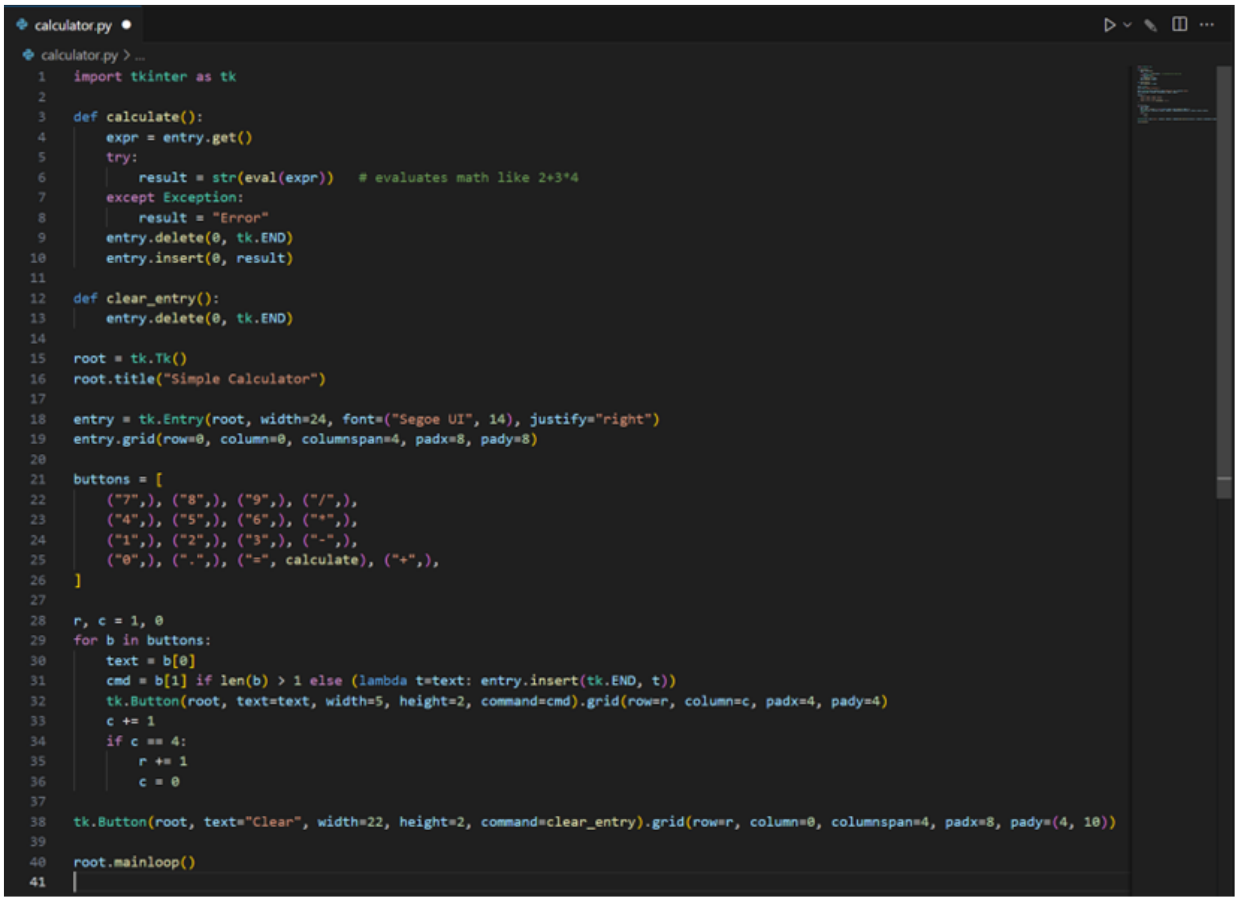
Open the Python-Calculator folder in your explorer and hover the mouse over the folder, then select new file and name it calculator.py



Copy and paste this code inside:

```
import tkinter as tkdef calculate():    expr = entry.get()    try:
result = str(eval(expr))   # evaluates math like 2+3*4    except
Exception:        result = "Error"    entry.delete(0,
tk.END)    entry.insert(0, result)def clear_entry():    entry.delete(0,
tk.END)root = tk.Tk()root.title("Simple Calculator")entry = tk.Entry(root,
width=24, font=("Segoe UI", 14), justify="right")entry.grid(row=0, column=0,
columnspan=4, padx=8, pady=8)buttons = [    ("7",), ("8",), ("9",),
("/",),    ("4",), ("5",), ("6",), ("*",),    ("1",), ("2",), ("3",),
("-",),    ("0",), (".",), ("=", calculate), ("+",),]r, c = 1, 0for b in
```

```
buttons:    text = b[0]    cmd = b[1] if len(b) > 1 else (lambda t=text:
entry.insert(tk.END, t))    tk.Button(root, text=text, width=5, height=2,
command=cmd).grid(row=r, column=c, padx=4, pady=4)    c += 1    if c
== 4:        r += 1        c = 0tk.Button(root, text="Clear",
width=22, height=2, command=clear_entry).grid(row=r, column=0, columnspan=4,
padx=8, pady=(4, 10))root.mainloop()
```


calculator.py
calculator.py > ...
1 import tkinter as tk
2
3 def calculate():
4 expr = entry.get()
5 try:
6 result = str(eval(expr)) # evaluates math like 2+3*4
7 except Exception:
8 result = "Error"
9 entry.delete(0, tk.END)
10 entry.insert(0, result)
11
12 def clear_entry():
13 entry.delete(0, tk.END)
14
15 root = tk.Tk()
16 root.title("Simple Calculator")
17
18 entry = tk.Entry(root, width=24, font=("Segoe UI", 14), justify="right")
19 entry.grid(row=0, column=0, columnspan=4, padx=8, pady=8)
20
21 buttons = [
22 ("7",), ("8",), ("9",), ("/",),
23 ("4",), ("5",), ("6",), ("*",),
24 ("1",), ("2",), ("3",), ("-",),
25 ("0",), (".",), ("=", calculate), ("+",),
26]
27
28 r, c = 1, 0
29 for b in buttons:
30 text = b[0]
31 cmd = b[1] if len(b) > 1 else (lambda t=text: entry.insert(tk.END, t))
32 tk.Button(root, text=text, width=5, height=2, command=cmd).grid(row=r, column=c, padx=4, pady=4)
33 c += 1
34 if c == 4:
35 r += 1
36 c = 0
37
38 tk.Button(root, text="Clear", width=22, height=2, command=clear_entry).grid(row=r, column=0, columnspan=4, padx=8, pady=(4, 10))
39
40 root.mainloop()
41

Run the App

At the top of VS Code, find the Run Python Button (triangle) click the down arrow button and select Run Python File.

Open Terminal → New Terminal.

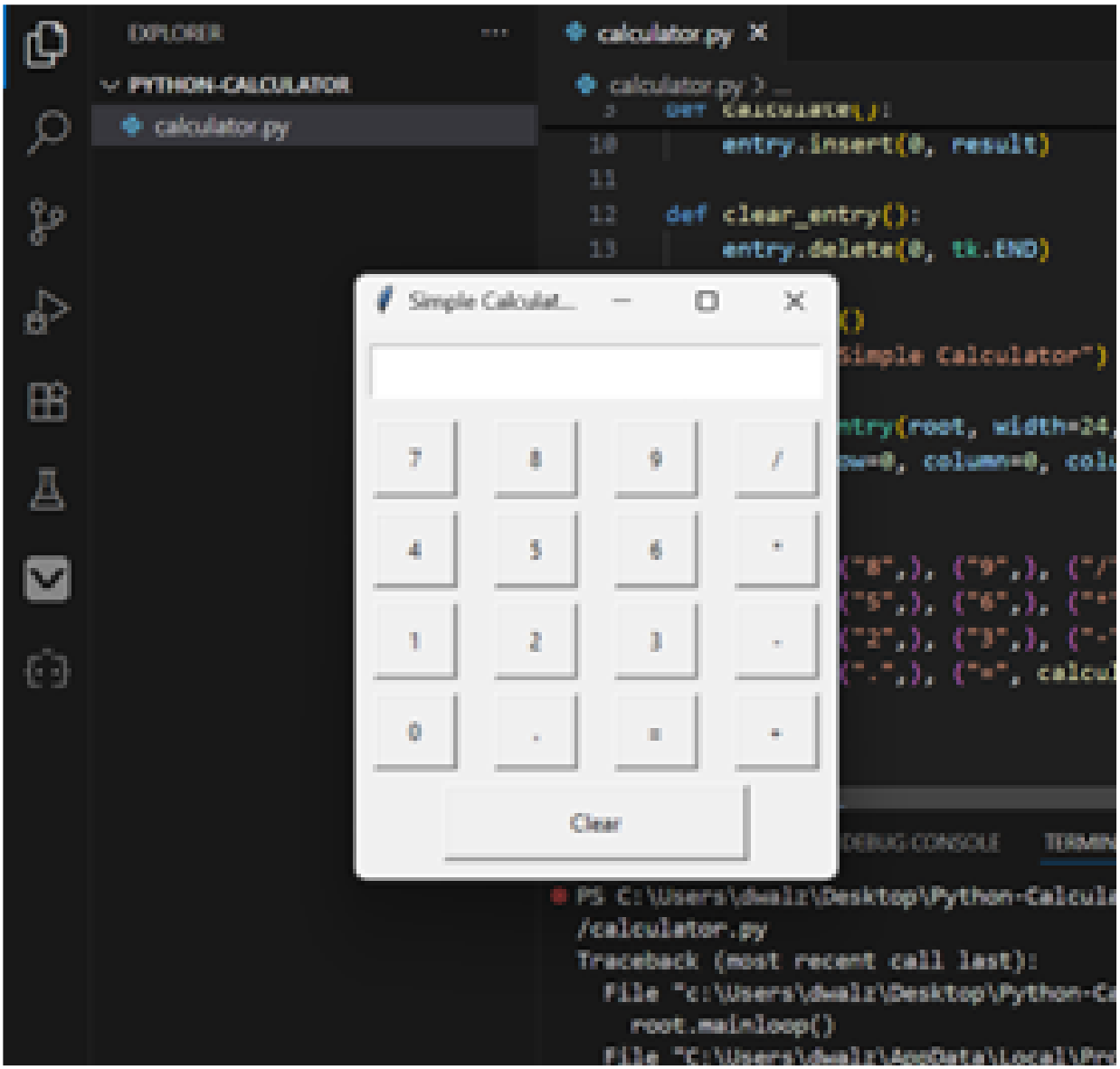
Type:

```
python calculator.py
```

A small calculator window will pop up!

• If you don't see it minimize your VS Code and check your desk top or run the file again.

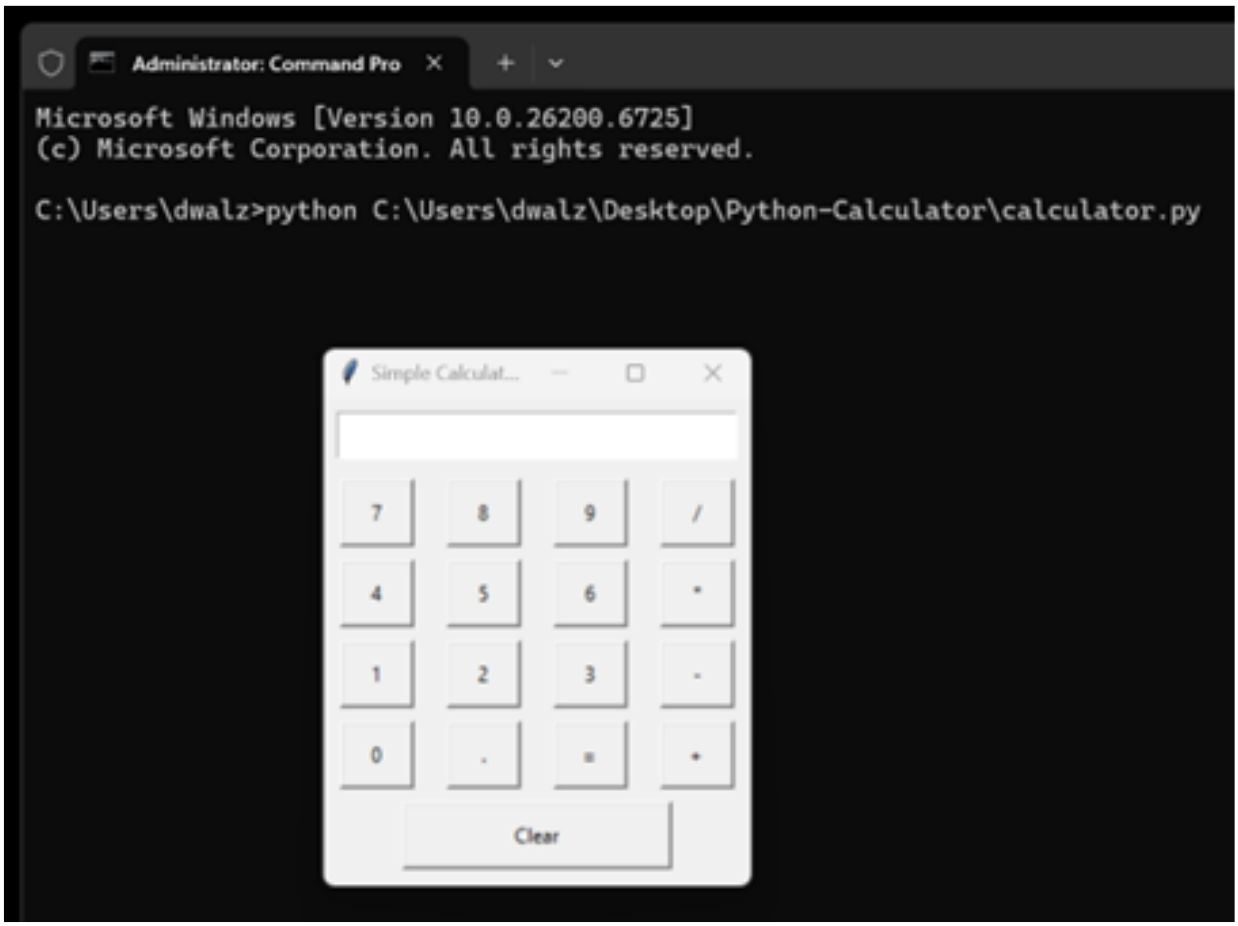
• Test the buttons and functions to see if its working.



Would you like to run the program without opening VS Code? Try this:

• Open the terminal; (Windows Key, type cmd, Open Terminal → New Terminal, Type: python C:\Users\dwalz\Desktop\Python-Calculator\calculator.py

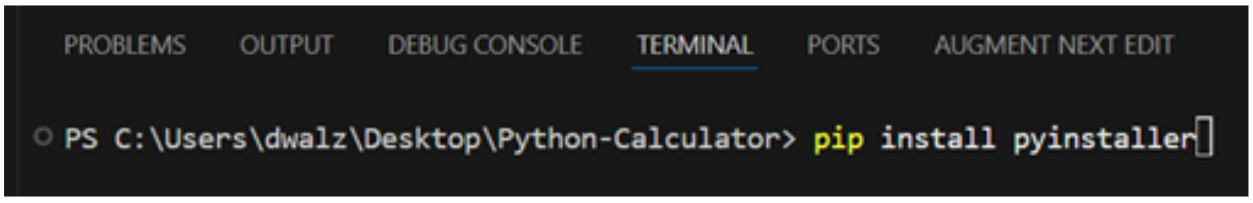
• Find the folder and copy the address as text to get the exact location. Above you can see my username, project folder, and file (it says .py because it's a python file).

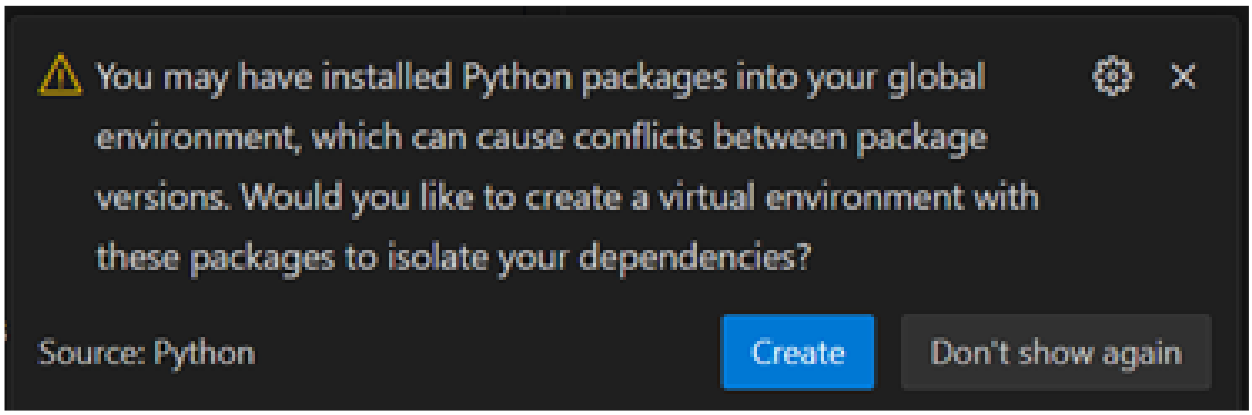


Package Your App as a Windows EXE

In the VS Code terminal, type:

```
pip install pyinstaller
```



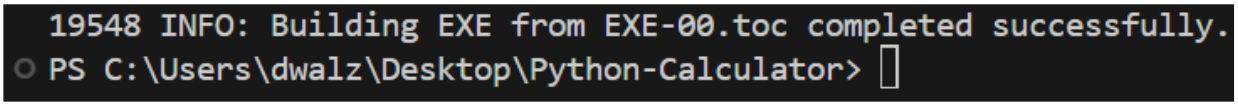

You may have installed Python packages into your global environment, which can cause conflicts between package versions. Would you like to create a virtual environment with these packages to isolate your dependencies?
Source: Python
Create
Don't show again

- For this message pop-up click create and at the top select the python environment at the bottom of the list.
- Here's the result pop-up message: The following environment is selected: ~\Desktop\Python-Calculator\.venv\Scripts\python.exe

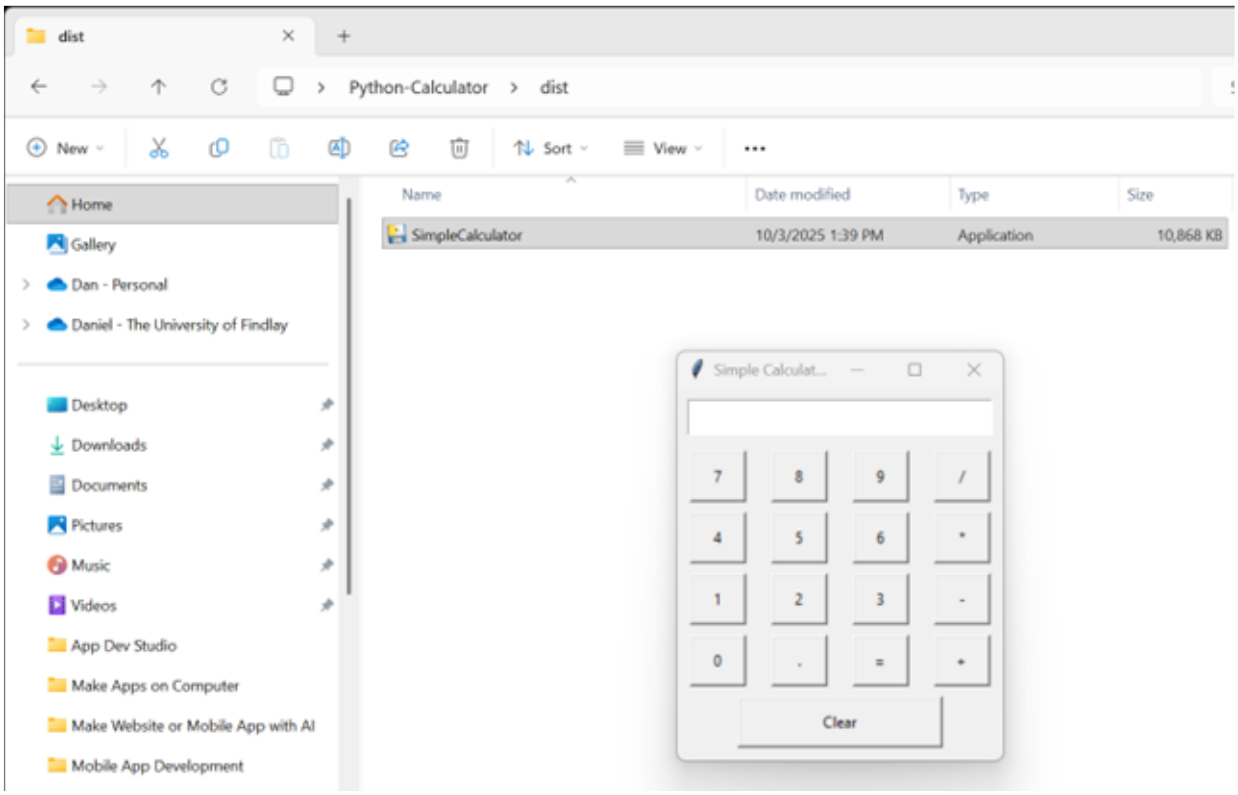
After it installs, in the terminal type:

```
pyinstaller --onefile --windowed --name SimpleCalculator calculator.py
```

When it finishes, look in the new dist folder → there's a file called SimpleCalculator.exe.



Double-click that file, and your calculator runs like any other app.



- Now you can copy that application file and share it with anyone or to any windows computer.
- Click File and Save your progress. Then click open folder and create another project folder like Stopwatch and generate some python code for it and make it the same way.
- To celebrate your work, save your program to a flashdrive and share it with others.

If you made it this far, you are well on your way to being able to build and deploy software on many different platforms. You did this the same way professional developers work, by breaking a large goal into smaller steps, installing the right tools, running commands carefully, testing often, and fixing problems one at a time. It can feel overwhelming at first, but each new project makes the workflow more familiar, and before long the commands and troubleshooting habits start to feel like second nature.

Why Python and VS Code Are Good Starting Points

Python is a great language for this stage of learning because it is readable, flexible, and widely used in both school and industry. With Python you can create desktop apps, automate repetitive computer tasks, process files and spreadsheets, build web servers and APIs, analyze data, create charts and reports, control hardware like Arduino or Raspberry Pi, and even work with machine learning and AI tools. That wide range of possibilities is one reason Python is so common in high school and university computer science courses. It lets students focus on core programming skills like variables, functions, loops, and problem solving while still building projects that feel useful and real.

Visual Studio Code also supports that same goal. VS Code is a free code editor made by Microsoft that has become a standard starting point for Windows development because it is fast, beginner friendly, and powerful enough for professional work. It supports many languages, connects easily to tools like Python and Git, and becomes more capable as students add extensions over time. As you continue learning, you can use this same workflow to build larger apps,

explore new frameworks, and publish projects with confidence because you already understand the process of turning code into something that runs for real users.

Chapter 5 Make a Windows App in Visual Studio Code Rubric

This chapter focused on creating a Windows calculator app in Visual Studio Code, running the app locally, packaging it as an executable file, testing the final program, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	Python and Visual Studio Code are installed or prepared. A project folder is created, clearly named, opened correctly, and organized with the required Python file.
Code and App Build	5	The calculator app is created with a usable interface, display area, number buttons, operation buttons, clear function, and equals function.
App Functionality	5	The calculator runs successfully, responds to button presses, displays input correctly, clears when needed, and produces accurate results for basic calculations.
Testing and Packaging	5	The app is tested in Visual Studio Code and packaged as a Windows executable file. Errors, layout issues, or functionality problems are identified and revised.
Evidence of Work	5	Screenshots, Python file evidence, terminal output, packaged executable proof, or other project evidence are submitted.
Reflection	5	A short explanation describes the Windows app, how it was created and tested, and relevant computer science vocabulary such as Python, file, function, variable, interface, terminal, package, executable, testing, or debugging.
Total	**30**	

Submission Checklist

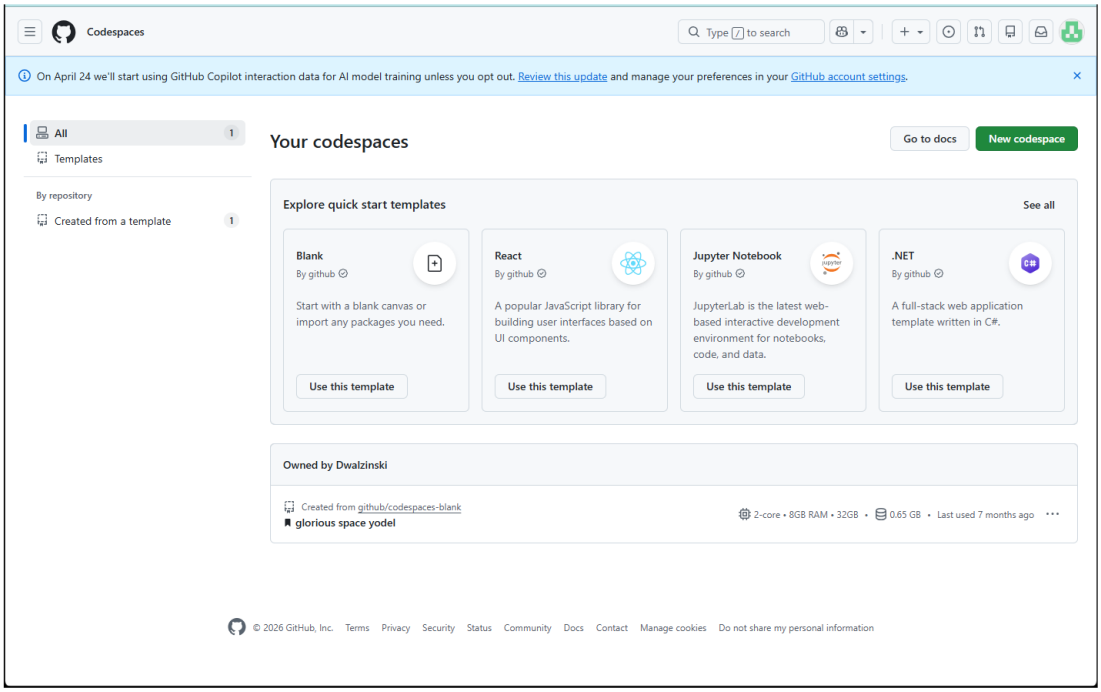
Requirement	Check
Python prepared or installed	☐
Visual Studio Code opened	☐
Project folder clearly named	☐
Python calculator file created	☐
Calculator interface completed	☐
Calculator tested with at least three examples	☐
At least one error, layout issue, or functionality issue revised	☐
Windows executable file created or attempted	☐
Screenshots or project proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the Windows calculator app created in Visual Studio Code. Explain how the app was run, tested, or packaged. Identify one error, design issue, or functionality problem that was tested and improved before final submission.*

6
Make App in GitHub Codespaces


Codespaces
Type / to search
On April 24 we'll start using GitHub Copilot interaction data for AI model training unless you opt out. Review this update and manage your preferences in your GitHub account settings.
All 1
Templates
By repository
Created from a template 1
Your codespaces
Go to docs
New codespace
Explore quick start templates
See all
Blank
By github
Start with a blank canvas or import any packages you need.
Use this template
React
By github
A popular JavaScript library for building user interfaces based on UI components.
Use this template
Jupyter Notebook
By github
JupyterLab is the latest web-based interactive development environment for notebooks, code, and data.
Use this template
.NET
By github
A full-stack web application template written in C#.
Use this template
Owned by Dwalzinski
Created from github/codespaces-blank
glorious space yodel
2-core • 8GB RAM • 32GB • 0.65 GB • Last used 7 months ago
© 2026 GitHub, Inc. Terms Privacy Security Status Community Docs Contact Manage cookies Do not share my personal information

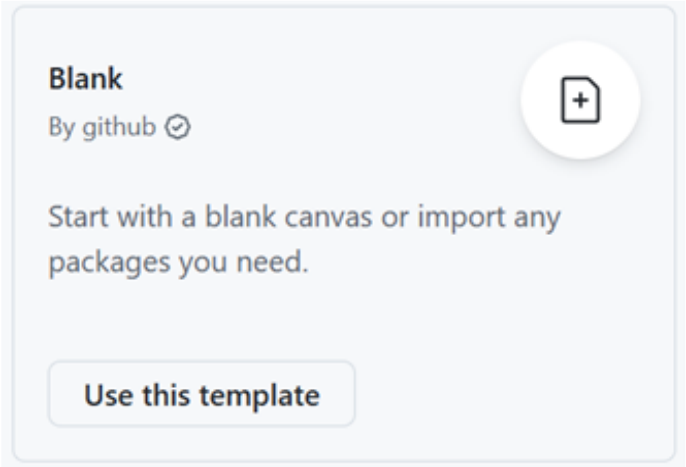
In this lesson you will build your first app using GitHub Codespaces, a cloud-based development environment that runs in your browser and works like Visual Studio Code. Instead of installing tools on your computer, you will create a Codespace from a blank template, generate a simple calculator using HTML, CSS, and JavaScript, and preview it instantly using the Live Server extension. By the end of the lesson, you will understand how browser-based development works, how to organize project files in a workspace, and how to run and view a web app in a real URL that you can test and share.

Start GitHub Codespaces

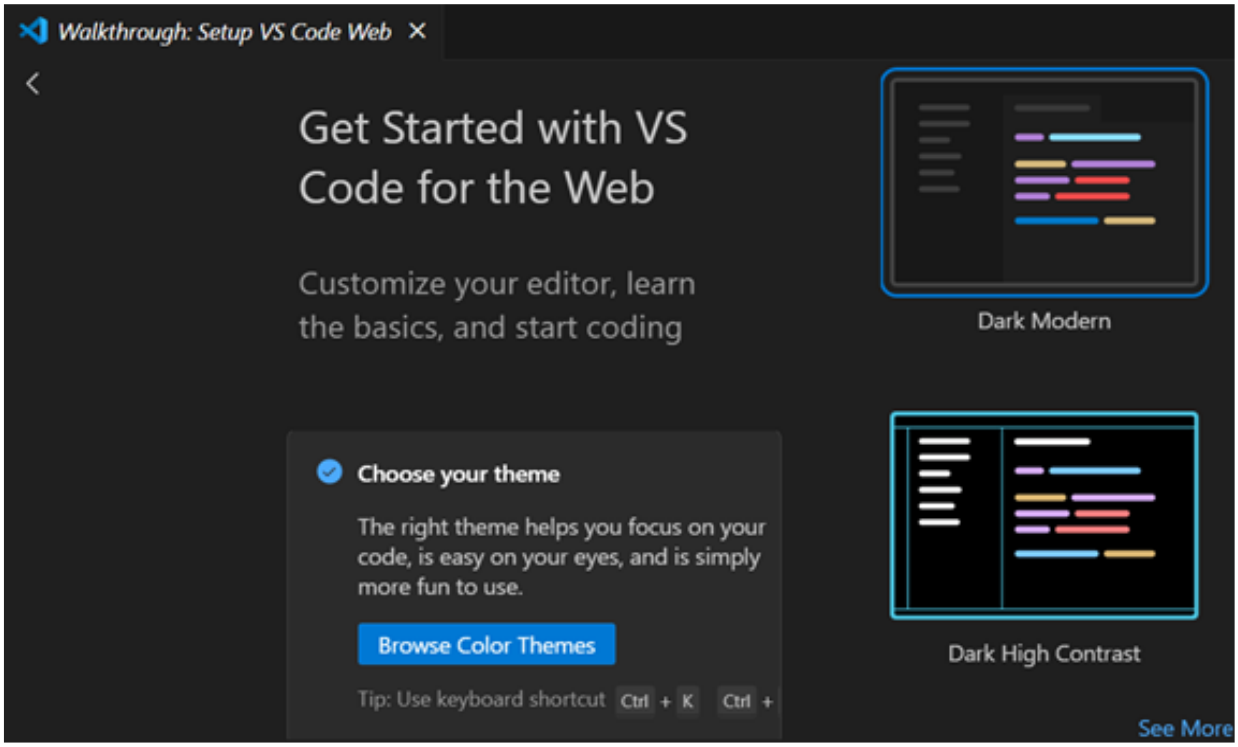
- Open the GitHub Codespaces page

Go to https://github.com/features/codespaces and click get started for free.

- Choose the Blank template
- Find the Blank template and click Use this template.

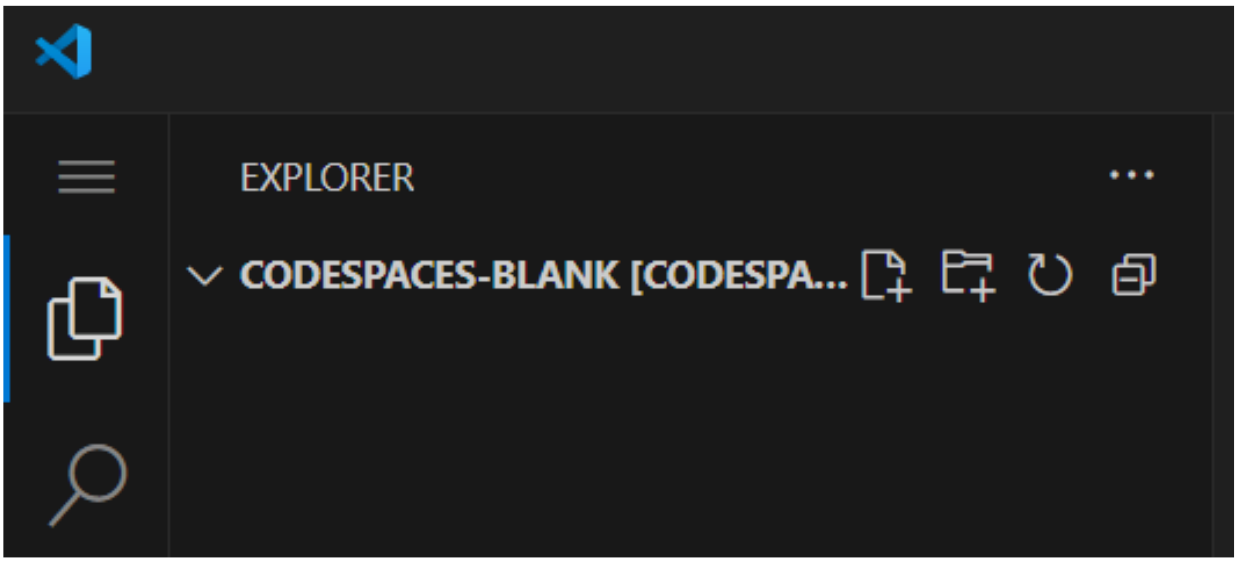


Wait a minute for it to load and if you want the same color scheme as me choose Dark modern and click the x to close the window.

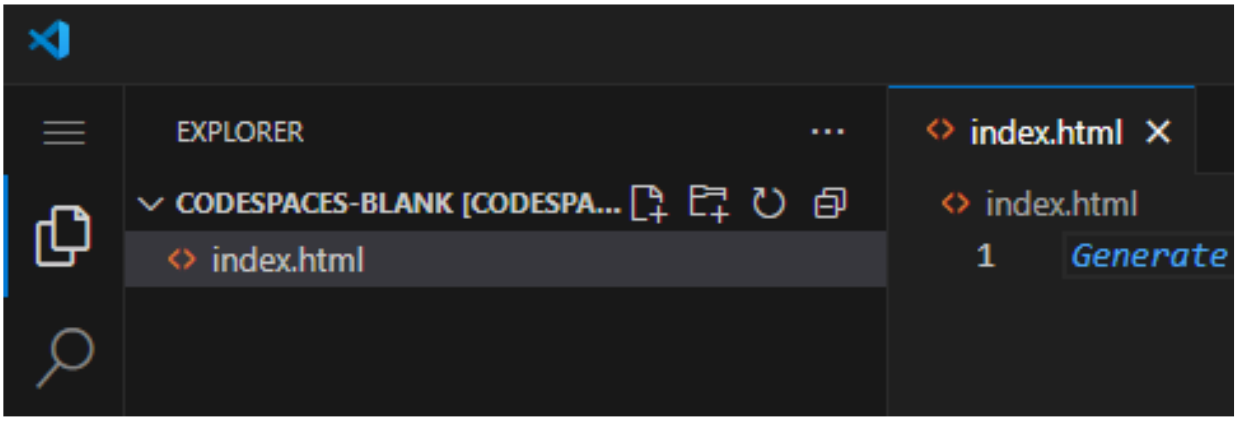

Walkthrough: Setup VS Code Web
Get Started with VS Code for the Web
Customize your editor, learn the basics, and start coding
Dark Modern
Dark High Contrast
Choose your theme
The right theme helps you focus on your code, is easy on your eyes, and is simply more fun to use.
Browse Color Themes
Tip: Use keyboard shortcut Ctrl + K Ctrl +
See More

Create your project files

- Open Explorer
- Click the top icon on the far left sidebar (looks like two stacked files).
- You should see: Open Editors, codespaces-blank (your folder), Outline, Timeline.



- Right-click your folder name (e.g., codespaces-blank) → New File → index.html



Generate the calculator code

Use an AI prompt or copy and paste the code from below.

Here's the prompt I used to get the code:

```
Inside github codespace create a code the lives entirely inside the index.html
file. Create code for a modern beautiful calculator
```

Open index.html and paste this code:

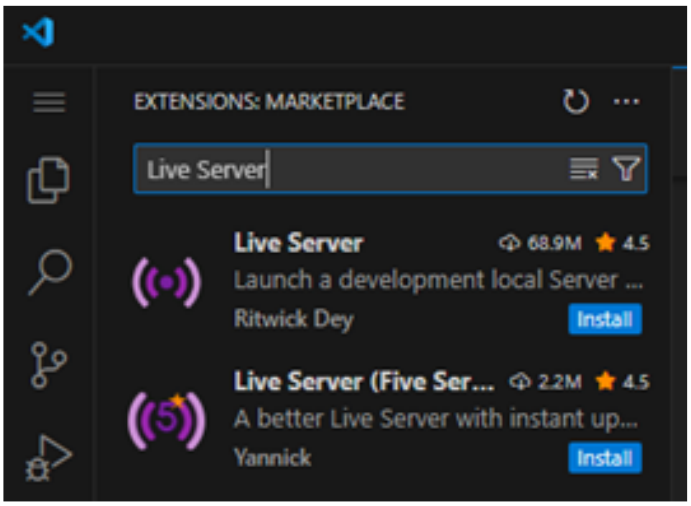
```
<!DOCTYPE html><html lang="en"><head>  <meta charset="UTF-8" />  <title>Modern Calculator</title>  <meta
name="viewport" content="width=device-width, initial-scale=1" />  <style>    :root{      --bg:
#0f1216;      --panel: #151a20;      --display: #0b0f14;      --text: #e8f0fe;      --muted:
#9fb3c8;      --btn: #232933;      --btn-hover: #2a3140;      --op: #394a7a;      --accent: #2e7d32;
--danger: #8b1c1c;      --shadow: 0 12px 30px rgba(0,0,0,.35);    }    .light{      --bg:#f6f8fb;
--panel:#ffffff;      --display:#eff3f9;      --text:#0f172a;      --muted:#475569;      --btn:#e8edf5;
--btn-hover:#dde5f2;      --op:#cfe0ff;      --accent:#2e7d32;      --danger:#c62828;      --shadow:
0 10px 24px rgba(0,0,0,.12);    }    *{box-sizing:border-box}    html,body{height:100%}    body{
```

```
margin:0;      font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, "Helvetica Neue",
Arial, "Apple Color Emoji","Segoe UI Emoji";      background: var(--bg);      color: var(--text);
display:grid;      place-items:center;      padding:24px;    }    .app{      width:min(380px,
92vw);      background: var(--panel);      border-radius: 20px;      box-shadow: var(--shadow);
overflow:hidden;      border:1px solid rgba(255,255,255,.06);    }    .topbar{      display:flex;
align-items:center; justify-content:space-between;      padding:14px 16px; background:transparent;    }
.brand{font-weight:800; letter-spacing:.4px}    .toggle{      display:inline-flex; align-items:center;
gap:8px;      font-size:13px; color:var(--muted);      user-select:none; cursor:pointer;    }    .switch{
width:46px; height:26px; background:var(--btn); border-radius:13px; position:relative; transition:.2s;
border:1px solid rgba(255,255,255,.05);    }    .switch::after{      content:""; position:absolute;
top:3px; left:3px; width:20px; height:20px; border-radius:50%; background:var(--display); transition:.2s;
box-shadow:0 2px 6px rgba(0,0,0,.25);    }    body.light .switch::after{ left:23px; }    .display{
background: var(--display);      min-height: 92px;      display:flex; align-items:flex-end;
justify-content:flex-end;      padding:18px;      font-variant-numeric: tabular-nums;    }    .expr{
color:var(--muted); font-size:14px; min-height:18px; width:100%; text-align:right; white-space:nowrap;
overflow:hidden; text-overflow:ellipsis;}    .value{ font-size:42px; font-weight:800; width:100%;
text-align:right; line-height:1.2; }    .keys{      padding:16px;      display:grid;
grid-template-columns: repeat(4, 1fr);      gap:10px;    }    button{      height:58px;      border:none;
border-radius:14px;      background: var(--btn);      color: var(--text);      font-size:18px;
font-weight:700;      cursor:pointer;      transition: transform .05s ease, background .15s
ease, opacity .15s ease;      box-shadow: inset 0 0 0 1px rgba(255,255,255,.04);    }
button:active{ transform:scale(.98); }    button:hover{ background: var(--btn-hover); }    .op{
background: var(--op); }    .eq{ background: var(--accent); }    .danger{ background: var(--danger);
}    .wide{ grid-column: span 2; }    .help{      text-align:center; color:var(--muted);
font-size:12px; padding:10px 16px 18px;    }    kbd{      background:var(--display); border:1px
solid rgba(255,255,255,.1);      padding:2px 6px; border-radius:6px; font-family:ui-monospace,
SFMono-Regular, Menlo, Consolas, monospace; font-size:11px;    }  </style></head><body>  <div
class="app" role="application" aria-label="Calculator">    <div class="topbar">      <div
class="brand">Modern Calculator</div>      <label class="toggle" title="Toggle Light/Dark (also press
D)">        <span>Theme</span>        <span class="switch" id="themeSwitch" aria-hidden="true"></span>
</label>    </div>    <div class="display" aria-live="polite" aria-atomic="true">      <div
style="width:100%">       <div class="expr" id="expr"> </div>       <div class="value" id="value">0</div>
</div>    </div>    <div class="keys" aria-label="Keypad">      <!-- Row 1 -->      <button
class="danger" data-action="clear" aria-label="All Clear">AC</button>      <button data-action="back"
aria-label="Backspace">⌫</button>      <button data-key="%" aria-label="Percent">%</button>      <button
class="op" data-key="÷" aria-label="Divide">÷</button>      <!-- Row 2 -->      <button
data-key="7">7</button>      <button data-key="8">8</button>      <button data-key="9">9</button>
<button class="op" data-key="×" aria-label="Multiply">×</button>      <!-- Row 3 -->      <button
data-key="4">4</button>      <button data-key="5">5</button>      <button data-key="6">6</button>
<button class="op" data-key="-" aria-label="Minus">-</button>      <!-- Row 4 -->      <button
data-key="1">1</button>      <button data-key="2">2</button>      <button data-key="3">3</button>
<button class="op" data-key="+" aria-label="Plus">+</button>      <!-- Row 5 -->      <button
class="wide" data-key="0">0</button>      <button data-key="." aria-label="Decimal">.</button>      <button
class="eq" data-action="equals" aria-label="Equals">=</button>    </div>    <div class="help">      Tip:
you can also type on your keyboard — try <kbd>1</kbd><kbd>+</kbd><kbd>2</kbd><kbd>=</kbd>. Press
<kbd>D</kbd> for theme.    </div>  </div>  <script>    // ===== Utilities =====    const exprEl  =
document.getElementById('expr');   const valueEl = document.getElementById('value');   const themeSwitch
= document.getElementById('themeSwitch');    let expr = "";     // what the user is building    let
shown = "0";   // current display    let justEvaluated = false;    const isOp = (ch) => ["+", "-",
"×", "÷"].includes(ch);    const toJS = (s) => s.replace(/÷/g,"/").replace(/×/g,"*").replace(/-/g,"-");
function updateDisplay() {      exprEl.textContent = expr || " ";      valueEl.textContent = shown;    }
```

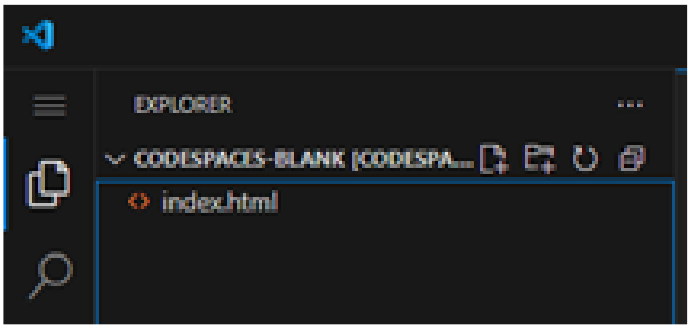
```
function clearAll() {      expr = "";      shown = "0";      justEvaluated = false;      updateDisplay();
}    function backspace() {      if (justEvaluated) { clearAll(); return; }      if (shown !== "Error" &&
shown.length > 1) {        shown = shown.slice(0, -1);      } else {        shown = "0";      }      //
Keep expr consistent      if (expr) expr = expr.slice(0, -1);      updateDisplay();    }    function
append(ch) {      if (shown === "Error") { clearAll(); }      if (justEvaluated && !isOp(ch)) { // start
new entry after equals unless it's an operator        expr = "";        shown = "0";        justEvaluated
= false;      }      // prevent two operators in a row      if (isOp(ch)) {        if (!expr && ch !== "-")
return;      // can't start with + × ÷       if (expr && isOp(expr.at(-1))) {         expr = expr.slice(0,
-1) + ch;      // replace last operator        } else {          expr += ch;        }        shown =
expr.split(/[+\-×÷]/).pop() || shown;       updateDisplay();       return;     }     // decimals: prevent
two dots in the same number segment      if (ch === ".") {        const segs = expr.split(/[+\-×÷]/);
const last = segs[segs.length - 1] || "";        if (last.includes(".")) return;      }      // percent:
convert like 25% -> (25/100)      if (ch === "%") {        // apply to last number only        const
m = expr.match(/(\d+(\.\d+)?)$/);        if (m) {          const start = m.index;          const num
= parseFloat(m[0]);          expr = expr.slice(0, start) + "(" + (num/100) + ")";          shown =
(num/100).toString();          updateDisplay();        }        return;      }      // normal digits or dot
if (expr === "" && ch === ".") expr = "0";      expr += ch;      shown = expr.split(/[+\-×÷]/).pop() ||
"0";      updateDisplay();    }    function evaluateExpr() {      if (!expr) return;      // don't end
with operator or dot      if (/[+\-×÷.]$/.test(expr)) return;      try {        const js = toJS(expr);
const result = Function(`"use strict"; return (${js})`)();       const num = (typeof result === "number"
&& Number.isFinite(result)) ? result : NaN;       if (Number.isNaN(num)) throw new Error("NaN");       //
round to max 10 decimals, trim trailing zeros        let out = parseFloat(num.toFixed(10)).toString();
shown = out;        expr = out; // allow chaining: 2+3= → then +4 → etc.        justEvaluated = true;
} catch {        shown = "Error";        justEvaluated = true;      }      updateDisplay();
}    // ===== Event wiring =====    document.querySelectorAll('[data-key]').forEach(btn=>{
btn.addEventListener('click', ()=>{        append(btn.getAttribute('data-key'));
});    });    document.querySelector('[data-action="equals"]').addEventListener('click',
evaluateExpr);    document.querySelector('[data-action="clear"]').addEventListener('click', clearAll);
document.querySelector('[data-action="back"]').addEventListener('click', backspace);    // Keyboard
support    window.addEventListener('keydown', (e)=>{      const k = e.key;      if (/\d/.test(k)) {
append(k); return; }     if (k === '.') { append('.'); return; }     if (k === '+' || k === '-') { append(k
=== '-' ? '-' : '+'); return; }     if (k === '*' || k === 'x') { append('×'); return; }     if (k === '/')
{ append('÷'); return; }     if (k === '%') { append('%'); return; }     if (k === 'Enter' || k === '=') {
evaluateExpr(); return; }     if (k === 'Backspace') { backspace(); return; }     if (k.toLowerCase() ===
'c') { clearAll(); return; }     if (k.toLowerCase() === 'd') { document.body.classList.toggle('light');
return; }    });    // Theme toggle    document.querySelector('.toggle').addEventListener('click',
()=>{      document.body.classList.toggle('light');    });    // Initial paint    updateDisplay();
</script></body></html>
```

Install the Live Server extension & view in browser

- On the left sidebar, click the Extensions icon, it looks like four squares.
- Search for 'Live Server' by Ritwick Dey and click Install.

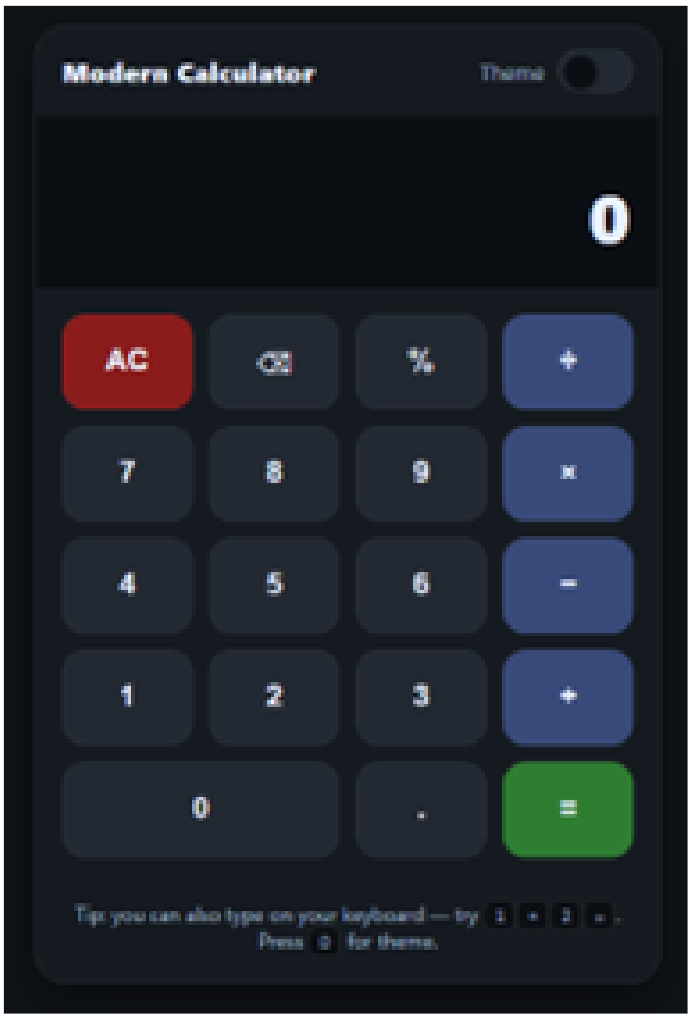


In Explorer, right-click index.html → Open with Live Server.



View your calculator in the browser

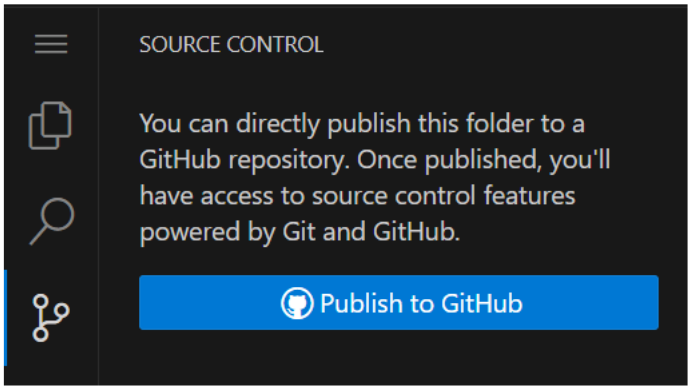
A new browser tab opens with your calculator. The URL may look like https://username-xxxx.app.github.dev.



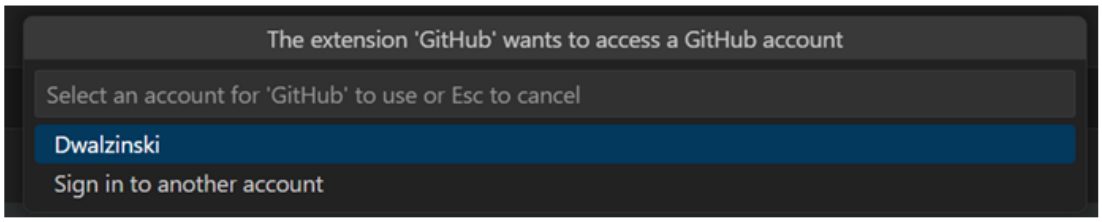
Publish to GitHub

Congratulations you successfully made an app. Do you want to let others use your app online? If so, follow the next steps.

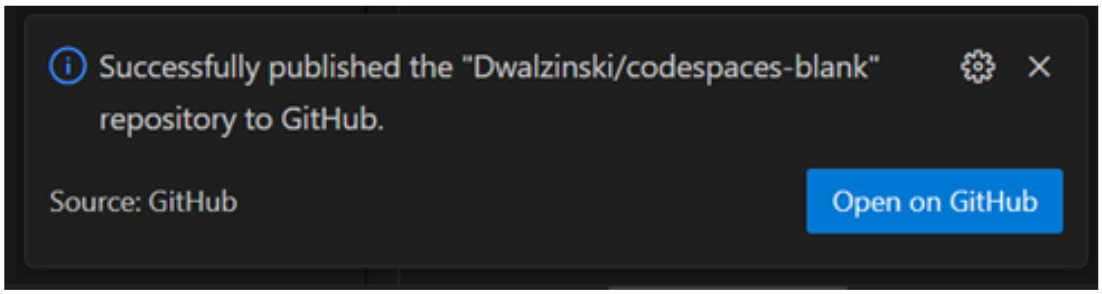
Find the 3 circles icon on the left and Publish to Github.


SOURCE CONTROL
You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by Git and GitHub.
Publish to GitHub

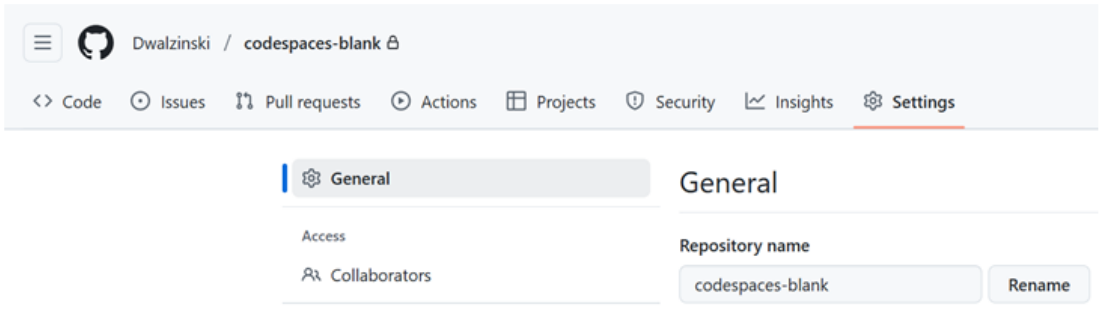
Choose your current account or create a new account. Tick the boxes and press ok.


The extension 'GitHub' wants to access a GitHub account
Select an account for 'GitHub' to use or Esc to cancel
Dwalzinski
Sign in to another account

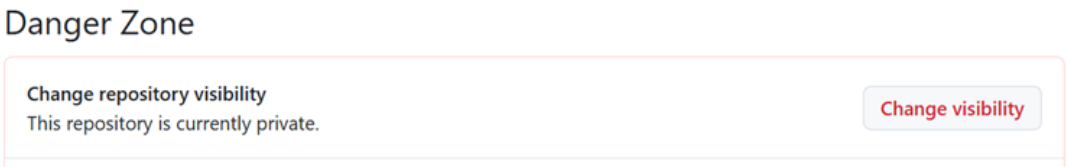
You will see this message in the bottom right corner of the screen.



Inside of github you can open the project and change the name in settings.

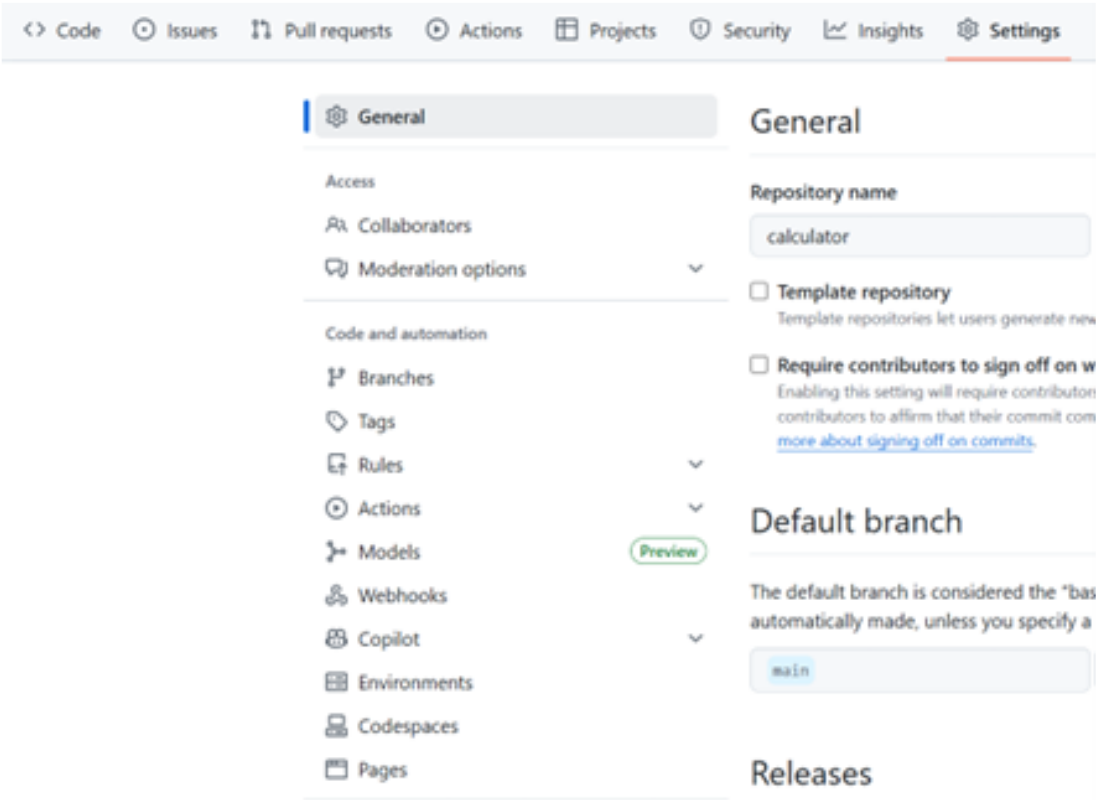


Inside of github scroll to Danger Zone and set to visibility and access to public.

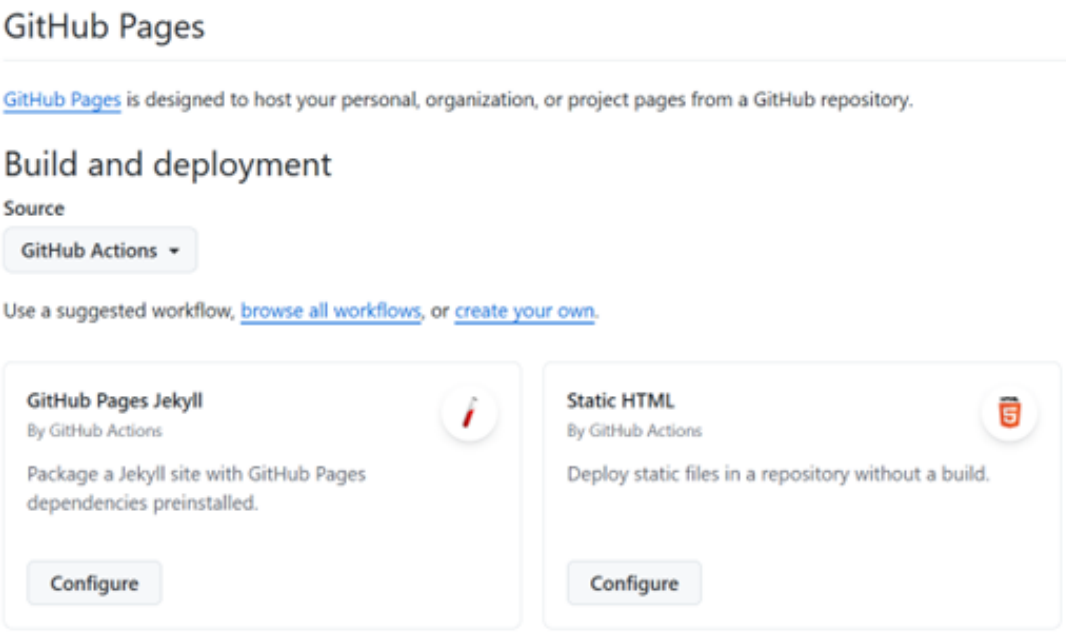


Create GitHub Pages

In the left panel scroll down to pages and create one.

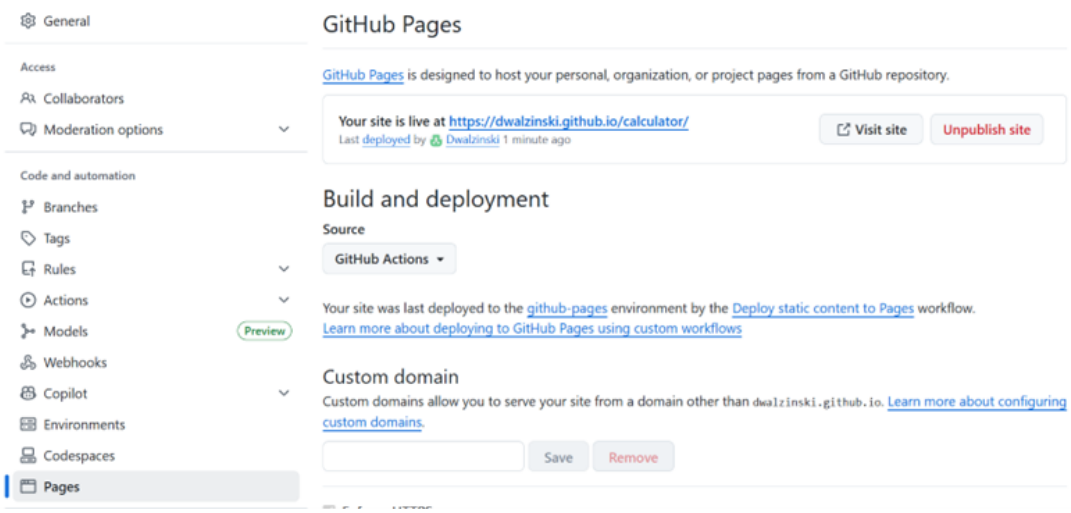


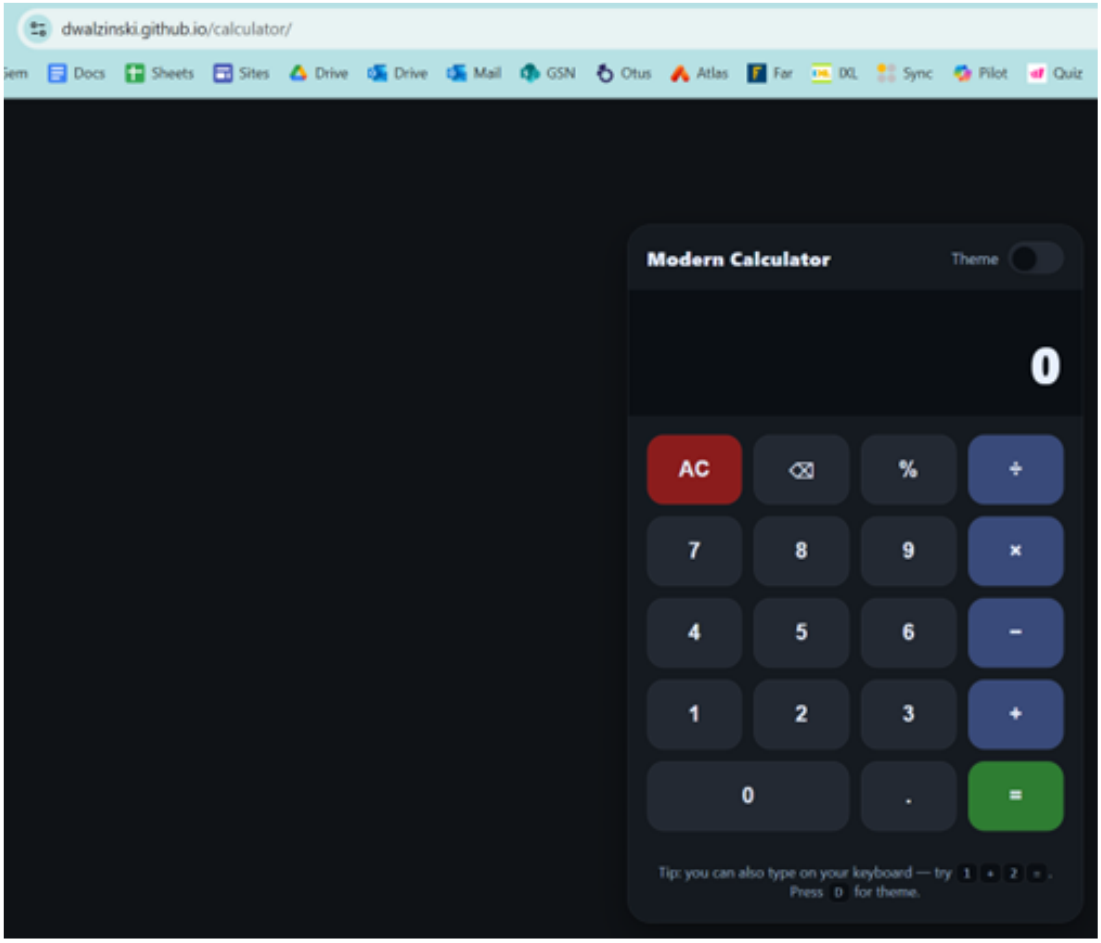
Choose static HTML


GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Build and deployment
Source
GitHub Actions
Use a suggested workflow, browse all workflows, or create your own.
GitHub Pages Jekyll
By GitHub Actions
Package a Jekyll site with GitHub Pages dependencies preinstalled.
Configure
Static HTML
By GitHub Actions
Deploy static files in a repository without a build.
Configure

Click commit changes

Go back to settings, then pages and you will find your URL there you can share with others.





You now have a working calculator running from a Codespace and previewed in your browser using Live Server, which means you successfully created a complete web app from scratch inside a cloud workspace. Before moving on, test every button and confirm the app opens correctly from the github.dev URL. If you want other people to use your project, publish it to GitHub, set the repository to public, and turn on GitHub Pages so your calculator becomes a shareable website link. If you finish early, duplicate the project and build a second one-page app (such as a stopwatch, flashcard page, or unit converter) using the same workflow: create files, generate code, test in the browser, then publish and share the final URL.

More about GitHub

GitHub is a website where developers store and manage coding projects in repositories. It works with Git, which is a version control tool that keeps a history of your changes over time. Instead of saving lots of different copies of the same file, you save your progress using commits, which are small checkpoints you can return to later. You can also create branches to try new ideas safely without breaking your main version, and use pull requests to review changes before combining them. This is why GitHub is important in the computer science community: it helps people collaborate, avoid losing work, and track exactly what changed and why.

For students, GitHub is also one of the best places to build a developer portfolio. You can keep a few strong projects public, add a clear README that explains what the project does and how to run it, and include screenshots or a short demo link. You can also publish a simple project website using GitHub Pages so anyone can open and test your work in a browser. Over time, your project history shows that you can plan, build, debug, and improve a program step by step, which is useful for classes, internships, and university applications.

Chapter 6 Make an App in GitHub Codespaces Rubric

This chapter focused on creating a web app in GitHub Codespaces, writing and organizing HTML, CSS, and JavaScript, testing the app in a browser preview, revising problems, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	A GitHub Codespaces project is created, clearly named, opened correctly, and organized with the required project files.
Code and App Build	5	The web app is created using organized HTML, CSS, and JavaScript. The interface includes a clear layout, display area, buttons, and required app elements.
App Functionality	5	The app opens in the browser preview, responds to user interaction, displays input correctly, and completes the required calculator or web app functions.
Testing and Revision	5	The app is tested in the browser preview. Errors, layout issues, broken buttons, or functionality problems are identified and revised.
Evidence of Work	5	Screenshots, project files, browser preview evidence, GitHub or Codespaces proof, or other project evidence are submitted.
Reflection	5	A short explanation describes the web app, how HTML, CSS, and JavaScript worked together, what was tested or revised, and relevant computer science vocabulary such as HTML, CSS, JavaScript, browser, file, function, event, variable, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
GitHub Codespaces project created	☐
Project clearly named and opened	☐
Required project files created	☐
HTML structure completed	☐
CSS styling completed	☐
JavaScript functionality completed	☐
App opened in browser preview	☐
At least one error, layout issue, or functionality issue revised	☐
Screenshots or project proof included	☐
Short reflection completed	☐

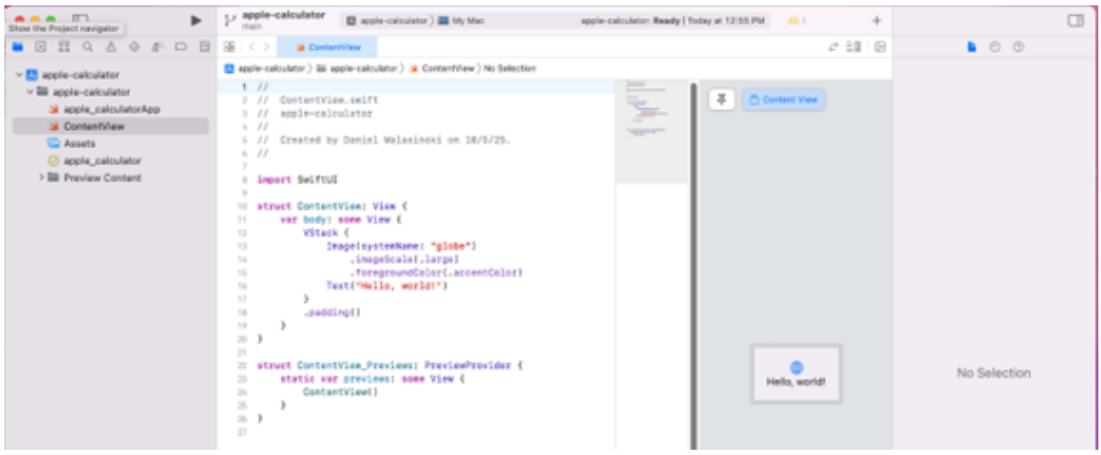
Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the web app created in GitHub Codespaces. Explain how HTML, CSS, and JavaScript worked together in the project. Identify one error, design issue, or functionality problem that was tested and improved before final submission.*

7

Make an App on Mac for Mac using Swift



In this lesson you will build a native macOS calculator using Swift and SwiftUI in Xcode, the standard development environment for Apple apps. You will create a new project, replace the starter code with your own calculator engine and user interface, then run the app and test it in a real Mac window. After it works, you will export a shareable Mac app, then create a professional app icon and rebuild so your project looks like a real product. By the end, you will understand the basic structure of an Apple app project and the complete workflow from project setup to a distributable app.

Install Xcode

- Open the App Store on your Mac or Macbook and download Xcode.

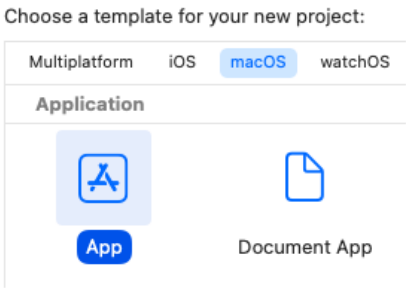
For older computers or operating systems you may need to download an older version of Xcode from apple developers website. https://developer.apple.com/download/all/?q=Xcode%2014.2

I determined the version I would need by entering in my computer information and operating system into an AI chat and discovered I will need to use Xcode 14.2 downloaded from the developers website above. Find this by clicking the Apple icon in the top left corner and selecting: About This Mac

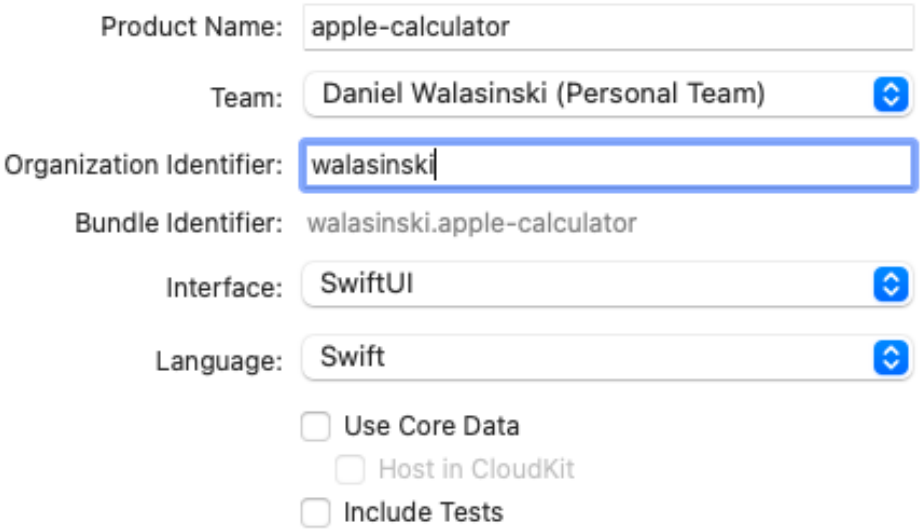
Open Xcode, Create a new Xcode project

Welcome to Xcode
Version 14.2 (14C18)
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch, or Apple TV.
Clone an existing project
Start working on something from a Git repository.
Open a project or file
Open an existing project or file on your Mac.

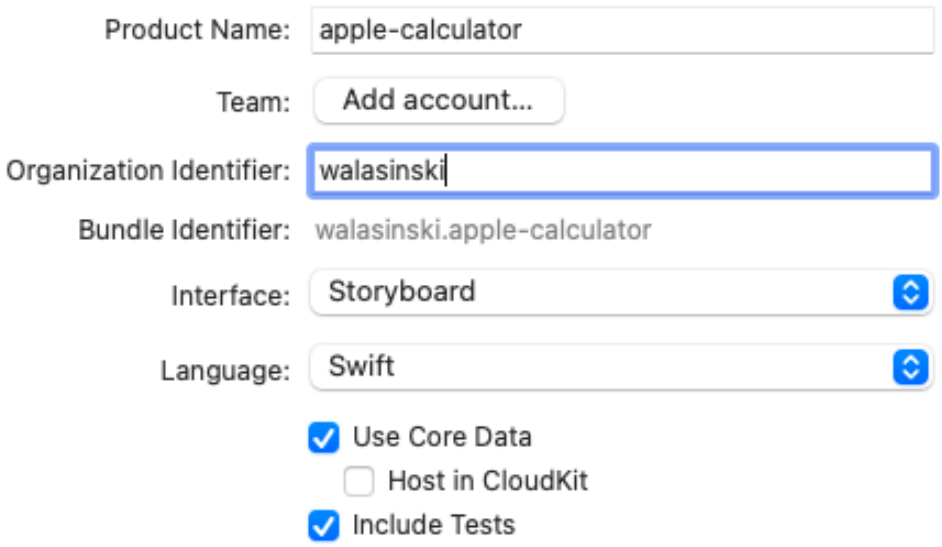
Choose App (macOS → App) → click Next.



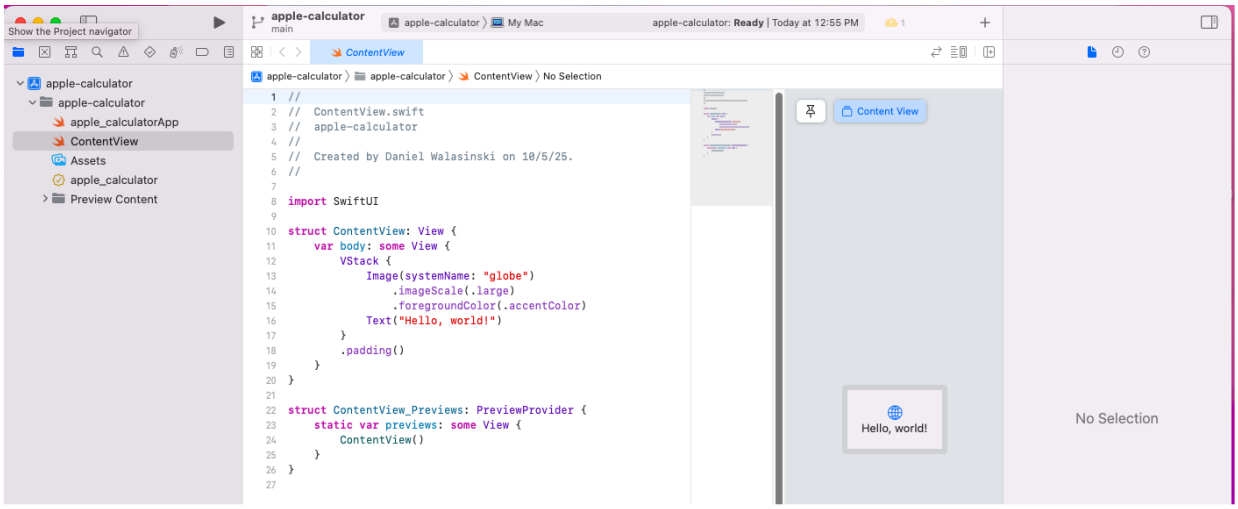
Name it: apple-calculator.5. Set Interface to SwiftUI



Set Language to Swift, use these checkmarked options.



Save it to your Desktop or create a folder inside your drive, one for all projects and one inside that for your calculator project.



Add the Swift Code

Remove the old code in the file and replace it with code for your calculator. Use the snippet below or generate code. Ensure the agent knows which version of Xcode you are running as well as the hierarchy of components you used.

The name in your explorer must match the code, here's mine:

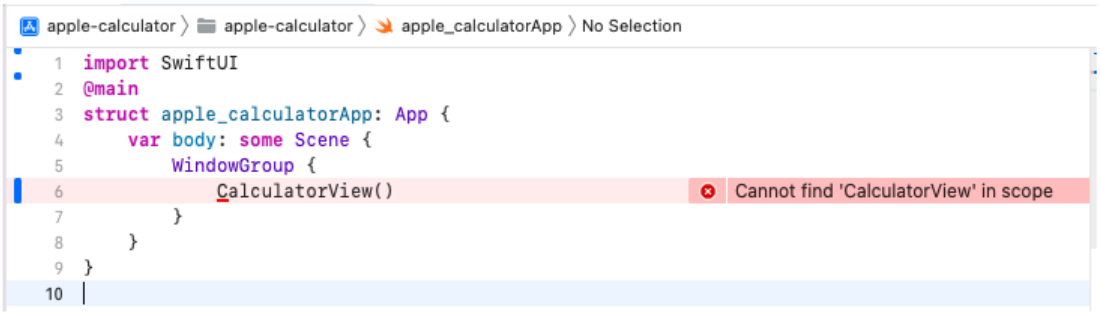
Replace your code with this setup

apple_calculatorApp.swift

Code:

```
import SwiftUI  @main struct apple_calculatorApp: App {     var body:
some Scene {         WindowGroup {             CalculatorView()
```

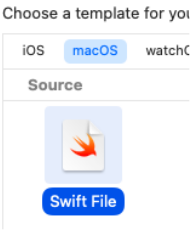
```
.frame(minWidth: 360, minHeight: 520) // nice default size on macOS          }
.windowStyle(.titleBar)     } }
```

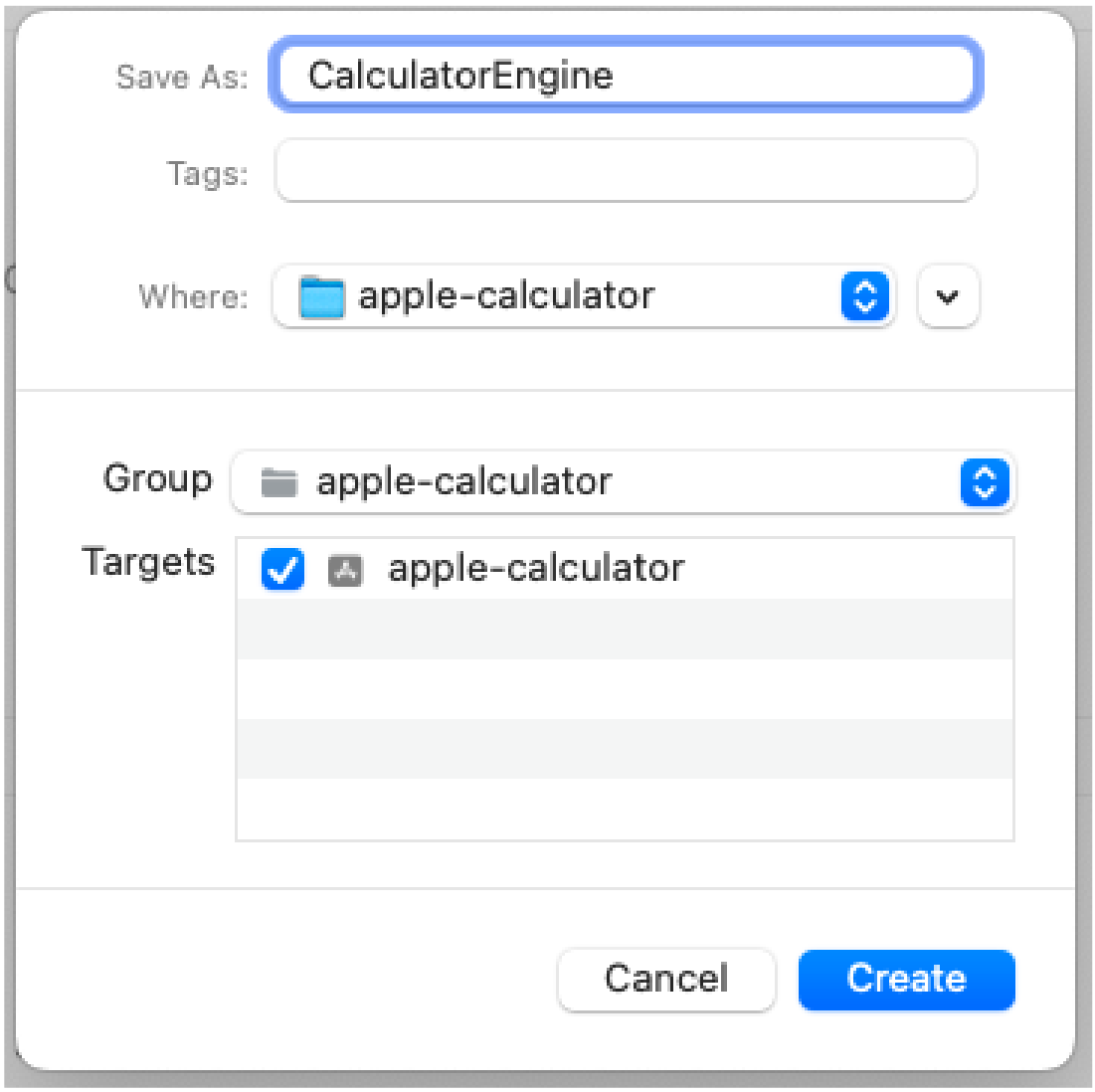


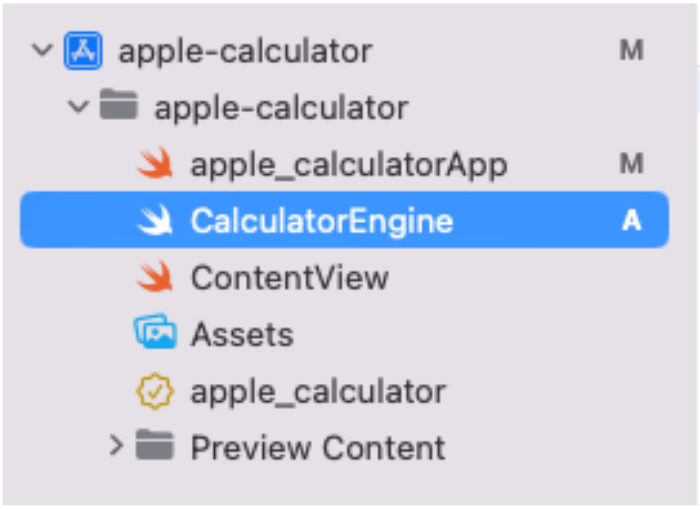
CalculatorEngine.swift

Create a new Swift file, right click the apple-calculator folder and create a new file.

Swift File → Name it: CalculatorEngine.swift







Drag the CalcualtorEngine file up to rest just below apple_calculatorApp

Use or generate this code in the new file:

Code:

```
import Foundation  enum Op { case add, subtract, multiply, divide } enum FuncKey { case clear, sign, percent } enum Key: Hashable {     case digit(String)    // "0"..."9", "00", "."     case op(Op)          // + − × ÷     case function(FuncKey) // AC, ±, %     case equals }  @MainActor final class CalculatorEngine: ObservableObject {     @Published var display: String = "0"      private var accumulator: Double? = nil     private var pendingOp: Op? = nil     private var isTyping: Bool = false      func tap(_ key: Key) {         switch key {         case .digit(let s):      appendDigit(s)         case .function(let f):   applyFunction(f)         case .op(let op):        applyOperation(op)         case .equals:            evaluate()         }     }      private func appendDigit(_ s: String) {         if !isTyping || display == "0" || display == "Error" {             display = (s == ".") ? "0." : s             isTyping = true         } else {             //  FIX: use && instead of the comma syntax             if s == "." && display.contains(".") { return }             display += s         }     }     private func applyFunction(_ f: FuncKey) {         switch f {         case .clear:             display = "0"; accumulator = nil; pendingOp = nil; isTyping = false         case .sign:             if display != "0", display != "Error" { if display.hasPrefix("-") { display.removeFirst() }                 else { display = "-" + display } }         case .percent:             if let v = Double(display) {                 display = format(v / 100.0)             }         }     }      private func applyOperation(_ op: Op) {         guard let rhs = Double(display) else { return }         if accumulator == nil {             accumulator = rhs         } else if let p = pendingOp {             accumulator = apply(p, accumulator!, rhs)         }         pendingOp = op         isTyping = false     }      private func evaluate() {         guard let p = pendingOp,               let lhs = accumulator,               let rhs = Double(display) else { return }          let result = apply(p, lhs, rhs)         if result.isFinite {             display = format(result)         } else {             display = "Error"         }         accumulator = nil         pendingOp = nil         isTyping = false     }      private func apply(_ op: Op, _ a: Double, _ b: Double) -> Double {         switch op {         case .add:      return a + b         case .subtract: return a - b         case .multiply: return a * b         case .divide:   return b == 0 ? .infinity : a / b         }     }      private func format(_ x: Double) -> String {         if x.rounded() == x { return String(Int(x)) }         // compact representation (about 10 significant digits)         return String(format: "%g", x)     } }
```

CalculatorView.swift

Code:

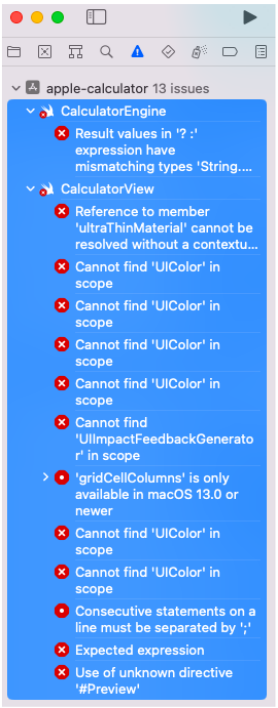
```
(Create another new Swift file called CalculatorView.swift) import SwiftUI  // MARK: - macOS-friendly theme private struct Theme {     static let corner: CGFloat = 18     static let displayCorner: CGFloat = 22     static let keyHeight: CGFloat = 64     static let spacing: CGFloat = 12      // Gradients & colors (AppKit-safe)     static let opGradient = LinearGradient(         colors: [Color.orange.opacity(0.95), Color.orange.opacity(0.75)],         startPoint: .topLeading, endPoint: .bottomTrailing     )     static let digitGradient = LinearGradient(         colors: [Color(nsColor: .controlBackgroundColor),                  Color(nsColor: .controlBackgroundColor).opacity(0.9)],         startPoint: .top, endPoint: .bottom     )     static let functionGradient = LinearGradient(         colors: [Color(nsColor: .windowBackgroundColor).opacity(0.9),                  Color(nsColor: .windowBackgroundColor).opacity(0.8)],         startPoint: .top, endPoint: .bottom     )     static let background = LinearGradient(         colors: [Color(nsColor: .windowBackgroundColor), Color(nsColor: .underPageBackgroundColor)],         startPoint: .top, endPoint: .bottom     )     static func outerShadow(_ scheme: ColorScheme) -> (Color, CGFloat, CGFloat) {         scheme == .dark         ? (Color.black.opacity(0.5), 14, 16)         : (Color.black.opacity(0.12), 10, 10)     } }  // MARK: - Haptic helper (macOS) private func performClickHaptic() {     #if os(macOS)     NSHapticFeedbackManager.defaultPerformer.perform(.generic, performanceTime: .now)     #endif }  // MARK: - Pressable style (depth + spring) private struct PressableStyle: ButtonStyle {     var bg: AnyShapeStyle     var fg: Color     var isOperator: Bool = false      func makeBody(configuration: Configuration) -> some
```

```
View {        configuration.label            .font(.system(size: 28, weight: .semibold, design: .rounded))
.foregroundStyle(fg)             .frame(height: Theme.keyHeight)             .frame(maxWidth:
.infinity)            .background(bg)            .clipShape(RoundedRectangle(cornerRadius: Theme.corner,
style: .continuous))          .overlay(              RoundedRectangle(cornerRadius: Theme.corner, style:
.continuous)                    .strokeBorder(Color.white.opacity(isOperator ? 0.12 : 0.08), lineWidth: 1)
.blendMode(.overlay)             )             .shadow(color: .black.opacity(configuration.isPressed ?
0.10 : 0.18),                     radius: configuration.isPressed ? 6 : 10,                     x:
0, y: configuration.isPressed ? 4 : 8)             .scaleEffect(configuration.isPressed ? 0.98 :
1)            .animation(.spring(response: 0.25, dampingFraction: 0.82), value: configuration.isPressed)
.onChange(of: configuration.isPressed) { pressed in                 if pressed { performClickHaptic()
}             }     } }  // MARK: - Calculator button private struct CalcButton: View {     let title:
String     let bg: AnyShapeStyle     let fg: Color     let action: () -> Void      init(title: String, bg:
some ShapeStyle, fg: Color, action: @escaping () -> Void) {         self.title = title         self.bg
= AnyShapeStyle(bg)         self.fg = fg         self.action = action     }      var body: some View
{        Button(title, action: action)            .buttonStyle(PressableStyle(bg: bg, fg: fg, isOperator:
isOperatorTitle))          .accessibilityLabel(accessibilityText)    }     private var isOperatorTitle:
Bool {         ["÷","×","-","+","="].contains(title)     }      private var accessibilityText: Text
{         switch title {         case "±": return Text("Toggle Sign")         case "AC": return
Text("All Clear")         case "÷": return Text("Divide")         case "×": return Text("Multiply")
case "-": return Text("Subtract")         case "+": return Text("Add")         case "=": return
Text("Equals")         default: return Text(title)         }     } }  // MARK: - Main View (LazyVGrid
for macOS compatibility) struct CalculatorView: View {     @Environment(\.colorScheme) private var scheme
@StateObject private var vm = CalculatorEngine()      // 4 equal columns     private let columns =
Array(repeating: GridItem(.flexible(), spacing: Theme.spacing), count: 4)      // Keys (no column-span;
macOS LazyVGrid doesn't support spanning—keeps it simple & clean)     private var keys: [[String]] = [
["AC","±","%","÷"],         ["7","8","9","×"],         ["4","5","6","-"],         ["1","2","3","+"],
["0",".","="] // last row has 3 keys; layout centers nicely     ]     var body: some View {        ZStack {
Theme.background.ignoresSafeArea()             VStack(spacing: Theme.spacing) {                displayCard
// Keypad                 VStack(spacing: Theme.spacing) {                     ForEach(0..<keys.count,
id: \.self) { row in                          LazyVGrid(columns: columns, spacing: Theme.spacing)
{                       ForEach(keys[row], id: \.self) { key in                             buttonView(for:
key)                             }                             // If row has only 3 keys,
add an invisible spacer cell to balance                             if keys[row].count == 3
{                             Color.clear                                 .frame(height: Theme.keyHeight)
}                       }                  }               }                .padding(.horizontal, 16)
.padding(.bottom, 8)             }             .padding(.top, 12)         }     }      // MARK:
- Display     private var displayCard: some View {         let (shadowColor, shadowRadius, shadowY) =
Theme.outerShadow(scheme)         return VStack {           Spacer(minLength: 0)           Text(vm.display)
.font(.system(size: 64, weight: .light, design: .rounded))                .monospacedDigit()
.lineLimit(1)                .minimumScaleFactor(0.28)                .frame(maxWidth: .infinity,
alignment: .trailing)                .foregroundStyle(.primary)                .padding(.horizontal, 18)
.padding(.bottom, 10)                .accessibilityIdentifier("display")         }         .frame(height:
128)         .background(             // Use a subtle translucent look that works on macOS (no Material
needed)           LinearGradient(colors: [               Color.white.opacity(scheme == .dark ? 0.06 : 0.28),
Color.white.opacity(0.04)            ], startPoint: .top, endPoint: .bottom)         )         .background(
RoundedRectangle(cornerRadius: Theme.displayCorner, style: .continuous)             .fill(Color(nsColor:
.windowBackgroundColor))         )         .clipShape(RoundedRectangle(cornerRadius: Theme.displayCorner,
style: .continuous))         .overlay(             RoundedRectangle(cornerRadius: Theme.displayCorner,
style: .continuous)                .stroke(LinearGradient(colors: [Color.white.opacity(0.22), .clear],
startPoint: .top, endPoint: .bottom), lineWidth: 1.1)                .blendMode(.plusLighter)         )
.shadow(color: shadowColor, radius: shadowRadius, x: 0, y: shadowY)         .padding(.horizontal, 16)     }
```

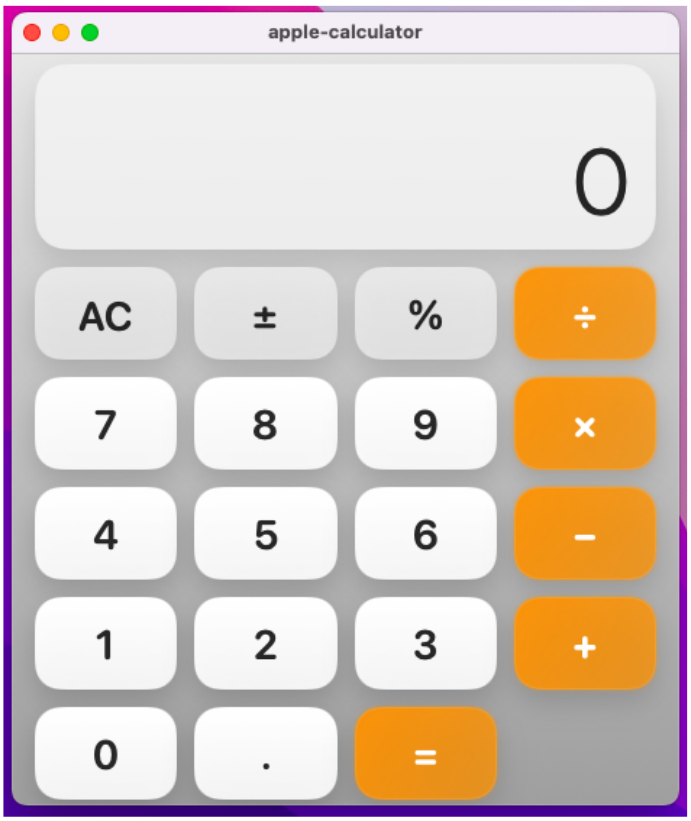
```
// MARK: - Button builder     @ViewBuilder     private func buttonView(for title: String) -> some View
{        switch title {        case "AC":            CalcButton(title: "AC", bg: Theme.functionGradient,
fg: .primary) {                vm.tap(.function(.clear))            }        case "±":
CalcButton(title: "±", bg: Theme.functionGradient, fg: .primary) {              vm.tap(.function(.sign))
}        case "%":             CalcButton(title: "%", bg: Theme.functionGradient, fg: .primary)
{               vm.tap(.function(.percent))           }        case "÷":            CalcButton(title:
"÷", bg: Theme.opGradient, fg: .white) {                vm.tap(.op(.divide))            }
.overlay(opGlow)        case "×":            CalcButton(title: "×", bg: Theme.opGradient, fg: .white)
{                vm.tap(.op(.multiply))            }            .overlay(opGlow)        case "-":
CalcButton(title: "-", bg: Theme.opGradient, fg: .white) {                vm.tap(.op(.subtract))
}            .overlay(opGlow)        case "+":            CalcButton(title: "+", bg: Theme.opGradient,
fg: .white) {               vm.tap(.op(.add))            }            .overlay(opGlow)        case "=":
CalcButton(title: "=", bg: Theme.opGradient, fg: .white) {                vm.tap(.equals)            }
.overlay(opGlow)        default:            // digits & dot            CalcButton(title: title, bg:
Theme.digitGradient, fg: .primary) {                vm.tap(.digit(title))            }        }     }
private var opGlow: some View {        RoundedRectangle(cornerRadius: Theme.corner, style: .continuous)
.stroke(Color.orange.opacity(0.32), lineWidth: 2)            .blur(radius: 1.2)     } }
```

Run the App

At the top left of Xcode, click the Run ▶ button. Your calculator opens in a Mac window, then test it out.


apple-calculator 13 issues
CalculatorEngine
Result values in '? :' expression have mismatching types 'String,...
CalculatorView
Reference to member 'ultraThinMaterial' cannot be resolved without a contextu...
Cannot find 'UIColor' in scope
Cannot find 'UIColor' in scope
Cannot find 'UIColor' in scope
Cannot find 'UIColor' in scope
Cannot find 'UIImpactFeedbackGenerator' in scope
'gridCellColumns' is only available in macOS 13.0 or newer
Cannot find 'UIColor' in scope
Cannot find 'UIColor' in scope
Consecutive statements on a line must be separated by ';'
Expected expression
Use of unknown directive '#Preview'

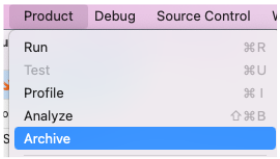
On my first try many errors came up, in that case I used shift click to copy and paste these to my AI agent to have them checked.



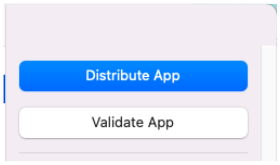
After making the adjustments here's the result, you can click and drag the corner to change the width of the calculator.

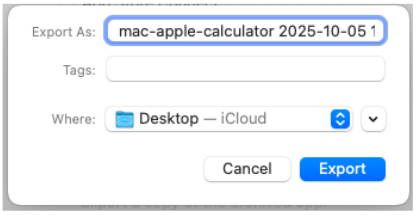
Export as a Mac App

In Xcode, go to Product → Archive.

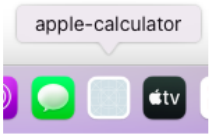


After it finishes, click Distribute App, choose copy and save to desktop or on your drive.





Open the file and drag the app to your app bar.



Now you got an app you can share with others.

Create an App Icon

Create an icon image to help others better understand your app

Make or export a square PNG image:

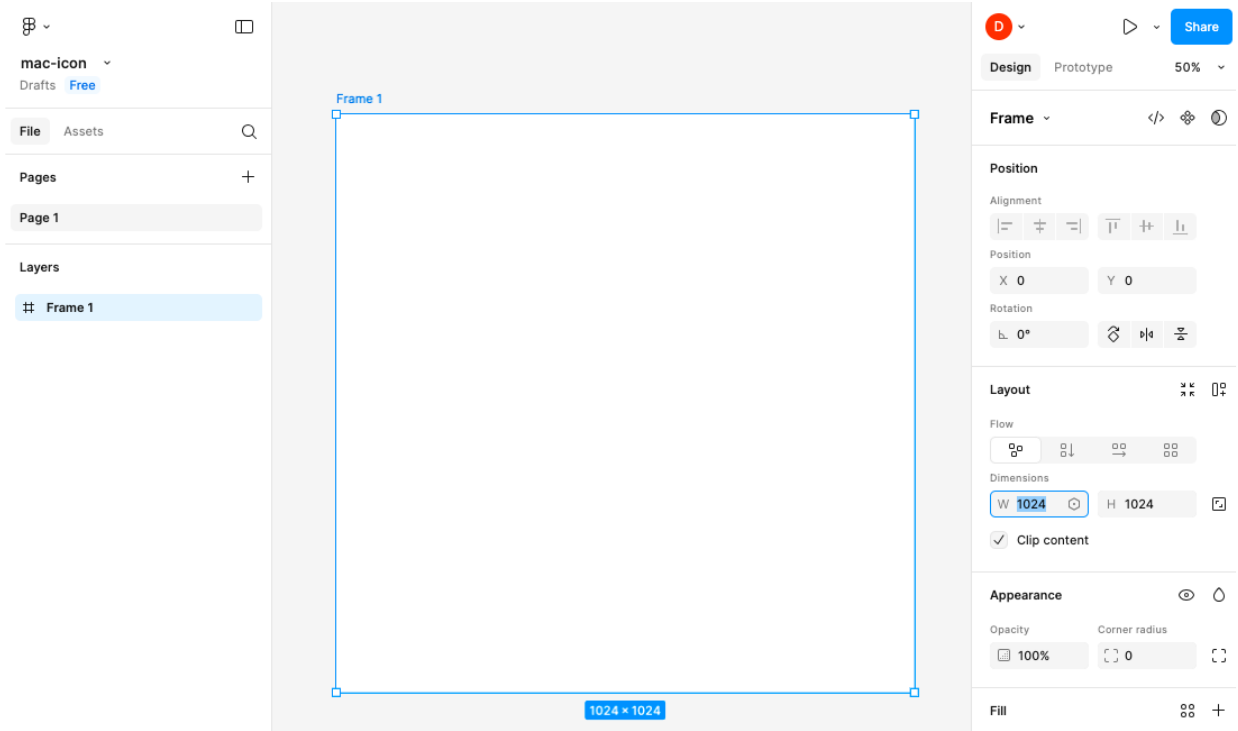
Size: 1024 × 1024 pixels

Background: transparent or solid, Apple recommends no rounded corners, macOS adds those automatically.

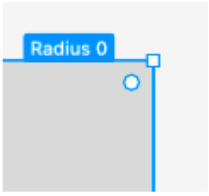
File format: PNG

Example filename: AppIcon.png

Go to Figma, click create new design, name is mac-icon, press F for frame or select it from the bottom menu, set the dimensions to 1024 × 1024 pixels. Hold control and use the mouse wheel to zoom in and out.



Add a square and cover the whole canvas you just placed. Then hover the cursor over the corner and click the corner circle to create a curved corner. Set the radius to 200px.



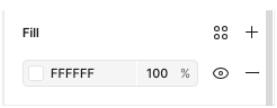
If you want, take a few different clipart icons, real icons, or inspirational images and upload them to an AI image generator to help it create something new and unique but guided while following your inspirations.

However this is what I did:

I tried to get a perfect square screenshot of my app using command, shift, 4 and used the visible px to try and make it square. Also I used a ruler and held it to the screen to square it as well.

I placed the screenshot on top of my rounded shape in Figma.

If you click on the canvas layer Frame 1 you will find that the color is set to white, you must delete it so that the background (the area outside of the curved corners) is transparent.

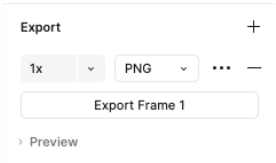


Next we need to change the color of the square we added to better match the screenshot of the calculator. On the left pane, choose the square and edit the color using the eyedropper to select a color from the calculator. If you used my

code for the calculator, you'll notice that the interface has a color gradient from the top (lighter) to bottom (darker), so add your own gradient to the square from the top to bottom, and use the eyedropper at the top and bottom of the calculator.

We can try other techniques to blend it more, but when the icon is scaled down it will also be less noticeable but for me I just added 50 to the corner radius of my calculator image.

Select Frame 1 and scroll to the bottom of the right pane to Export, export as PNG



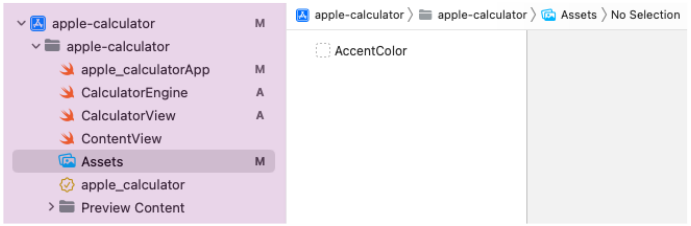
Open your icon and paste it inside a word document to verify that it's transparent outside the corners.

Generate Icon Sizes

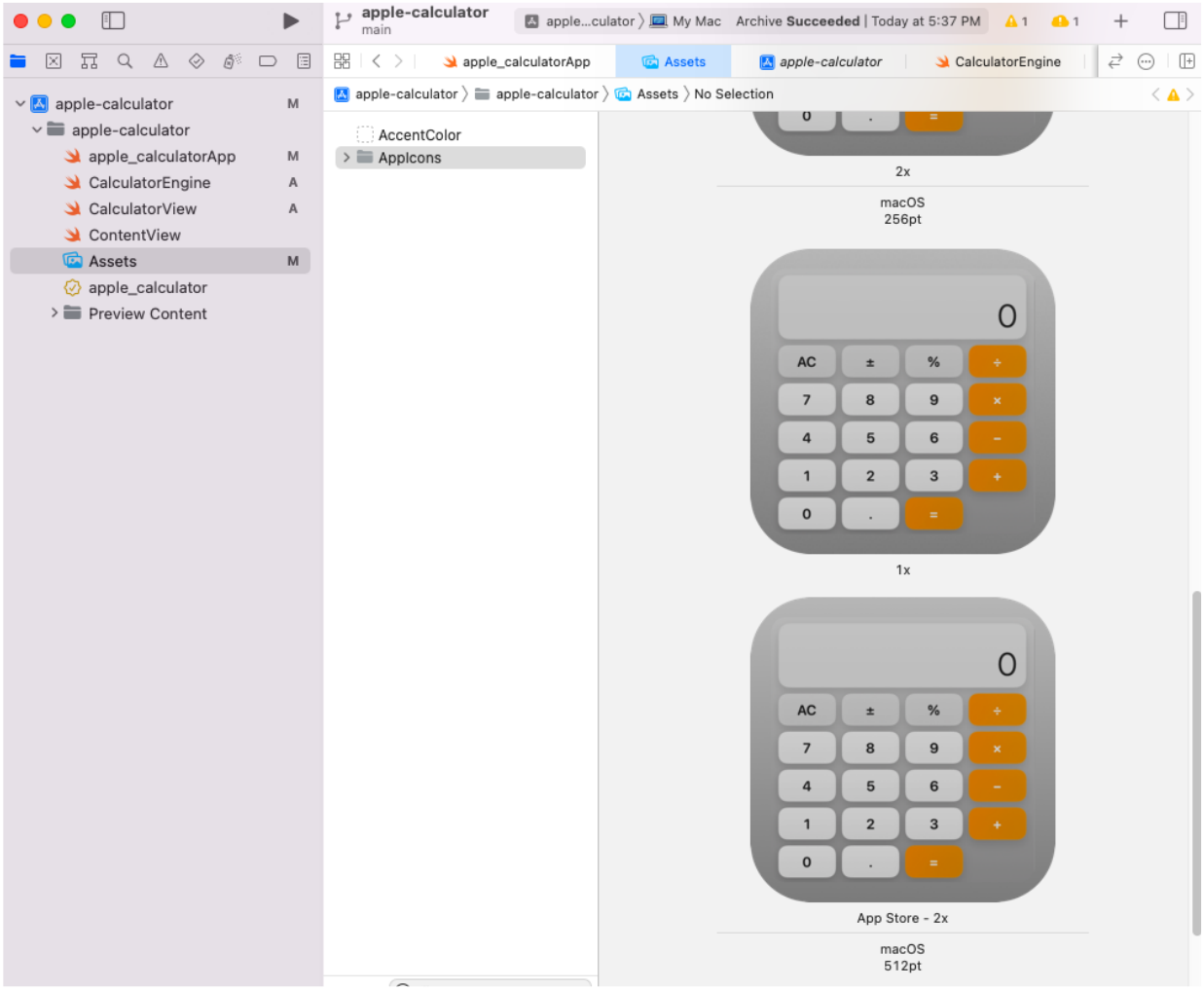
Mac apps use multiple resolutions. You can easily generate them from your 1024×1024 master image using one of these free tools or by doing it manually.

Go to https://realfavicongenerator.net/ Upload your image, scroll to the bottom click next, download your favicon package.

Go back inside of Xcode to your project, open Assets.

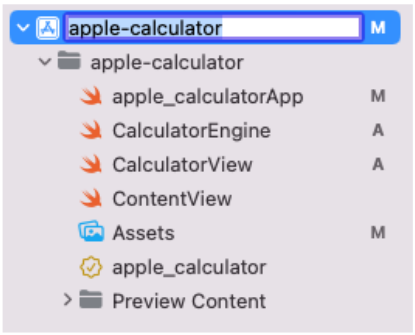


Open your Favicon folder, drag and drop the 512x512 size file in as well as your 1024x1024 Figma fila into the correct slots.



Confirm the icon in project settings (don't need to change anything)

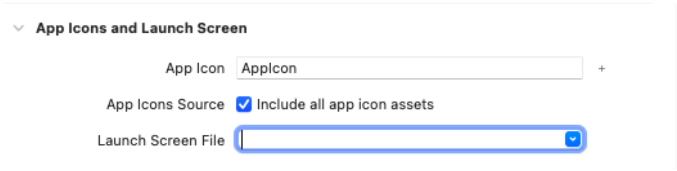
Click your project name (blue icon at the very top of the navigator).



In the Targets list, select apple_calculatorApp.

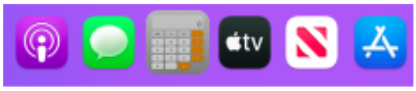
Go to the General tab.

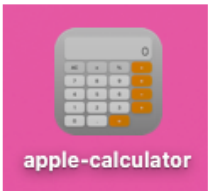
Under App Icons and Launch Images → App Icon, make sure it says AppIcon (the same name as your set).



Rebuild

Now In Xcode, go to Product → Archive, use same steps as before.





I can see my app icon now on the app bar after adding the new one.

Post Note: I think we could reduce the total size in Figma and use more transparent space to make the app perfectly sized matched with the standard mac apps.

Xcode and App Store Connect in the Apple Development Workflow

code is Apple's integrated development environment for building apps that run on iPhone, iPad, Mac, Apple Watch, Apple TV, and Apple Vision Pro. It organizes your work as a project with settings, signing information, and build targets, then lets you test your app quickly using Simulator for many Apple devices and system versions. When you are ready to share your app, Xcode can create an archived build and prepare it for beta testing or release, which is the same archive workflow you used in this lesson. Apple also updates submission requirements over time, and apps uploaded to App Store Connect must meet Apple's current minimum requirements, which is why Xcode is the standard tool for shipping Apple apps.

Xcode is not the only place you can write code, but it is the main tool used to run, package, and submit Apple apps. Developers often write Swift in other editors such as VS Code or Cursor, but they still rely on the Xcode toolchain on a Mac for building, testing, signing, and distributing. App Store Connect is the web portal used to upload builds, manage versions and releases, and invite testers through TestFlight before a public launch. Uploading builds is commonly done with Xcode, and Apple also supports upload tools like Transporter for certain workflows. For beginners, Apple also offers Swift Playgrounds as a learning-first option on Mac and iPad, while Xcode remains the primary path for full app projects and publishing.

Chapter 7 Make an App on Mac for Mac Using Swift Rubric

This chapter focused on creating a macOS app in Xcode using Swift, organizing project files, building the app interface, testing the app, revising problems, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	An Xcode macOS project is created, clearly named, opened correctly, and organized with the required Swift files and project settings.
Code and App Build	5	The app is created using Swift or SwiftUI with a usable interface, clear layout, required controls, and organized code.
App Functionality	5	The app runs successfully on Mac, responds to user interaction, displays information correctly, and completes the required app functions.
Testing and Revision	5	The app is tested in Xcode. Errors, layout issues, broken controls, or functionality problems are identified and revised.
Evidence of Work	5	Screenshots, Swift file evidence, Xcode preview or run evidence, app icon proof, exported app proof, or other project evidence are submitted.
Reflection	5	A short explanation describes the Mac app, how Swift or SwiftUI was used, what was tested or revised, and relevant computer science vocabulary such as Swift, Xcode, interface, view, state, function, variable, build, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
Xcode project created	☐
macOS app project clearly named	☐
Required Swift files created or edited	☐
App interface completed	☐
Required app features completed	☐
App tested in Xcode	☐
At least one error, layout issue, or functionality issue revised	☐
App icon or app appearance checked	☐
Screenshots or project proof included	☐
Short reflection completed	☐

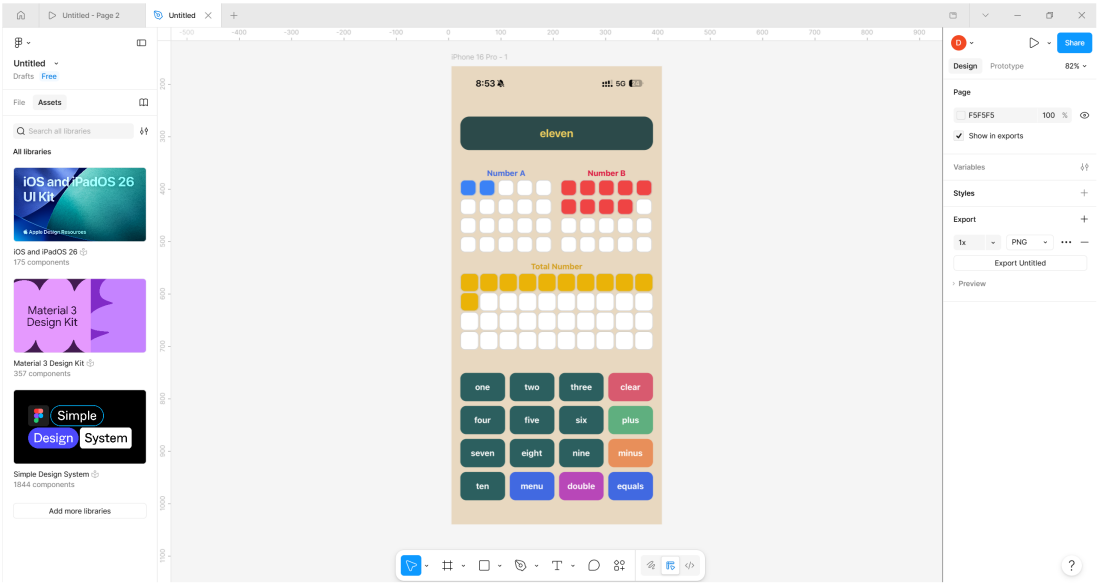
Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the Mac app created in Xcode. Explain how Swift or SwiftUI was used to build the app interface or functionality. Identify one error, design issue, or functionality problem that was tested and improved before final submission.*

8

Design App User Interface in Figma

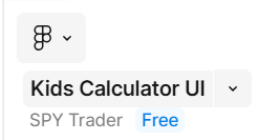


This lesson focuses on planning the visual design of a mobile app before writing code. You will create a complete calculator interface in Figma for children learning English math, including a text-based display, a solar panel detail, and a grid of colorful buttons. The steps emphasize core interface design techniques used in software projects: selecting an appropriate device frame, setting layout guides, using Auto Layout for consistent spacing, grouping elements into reusable structures, and exporting a final screen as an image that can be referenced during app development.

Lesson Goal

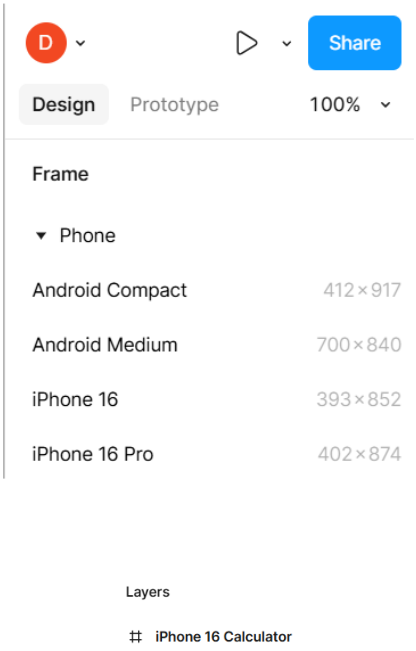
Design a playful calculator app for children learning English math. The calculator will use text instead of numbers ('one', 'two', 'plus', 'equals'), have a solar panel like a real calculator, include colored light-up blocks, and be designed for mobile devices.

- Go to figma.com or download the app and log in.
- Click + New Design File.
- Rename the file at the top center: Kids Calculator Visual UI.
- Press Enter.



Create the iPhone Frame

- Press F (Frame tool).
- From the right sidebar → Frame Presets → Phone → choose iPhone 16 Pro (402 × 874 px).
- Rename the frame to iPhone 16 Calculator.

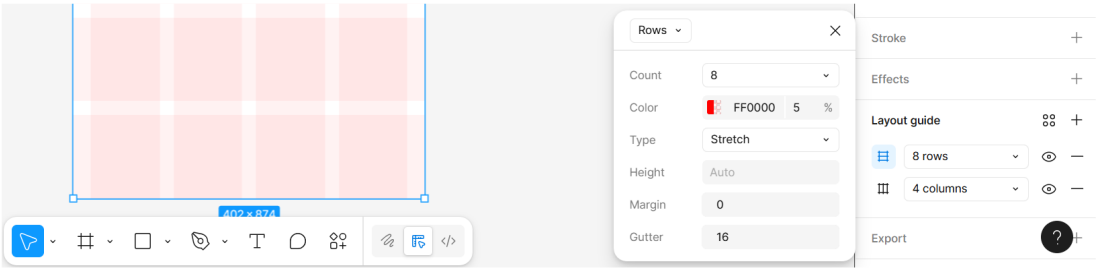


Set Up Guides

- Select your frame.
- In the right sidebar → Layout guide → +
- Change Grid → Columns.
- Settings:

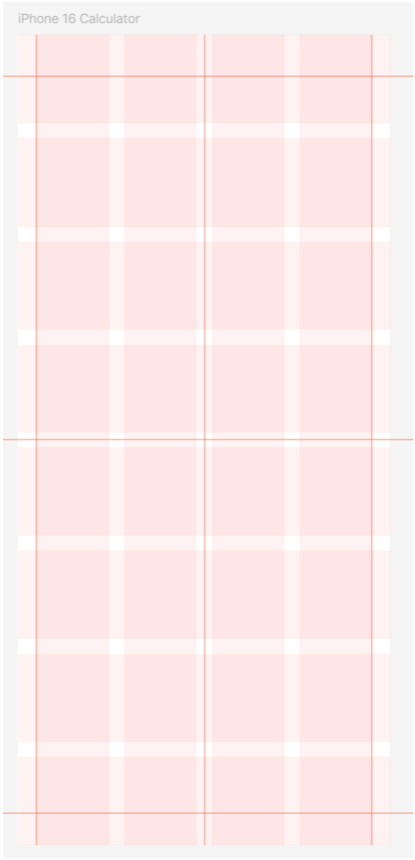
Type: Columns | Count: 4 | Gutter: 16 px | Margin: 20 px | Color Opacity: 10%

- Add a second grid by pressing + → Rows: Count = 8, Gutter = 16 px.



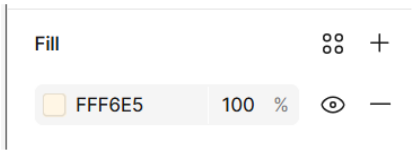
Press Shift + R to toggle rulers, then click and drag guidelines from the rulers to X = 201 px and Y = 437 px to create centerlines. Next, add perimeter guides at 44 px from the top, 34 px from the bottom (Y = 840), and 20 px from the left

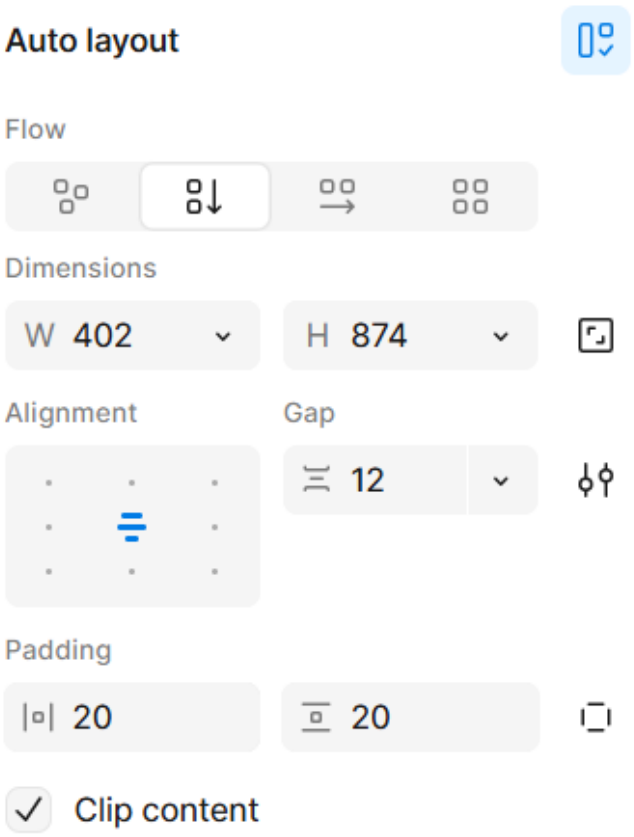
and right (X = 382) edges to define the safe area around the screen. Hold control and use the mouse wheel to zoom in and out.



Frame Background and Auto Layout

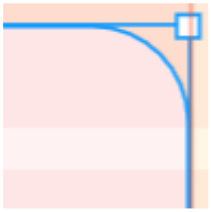
- Select frame → Fill → #FFF6E5.
- Press Shift + A to apply Auto Layout.
- In sidebar set: Direction: Vertical, Alignment: Center, Gap: 12 px, Padding: 24 px.



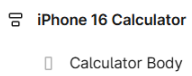


Calculator Body

- Press R → draw rounded rectangle (Width = 360 × Height = 760 px).
- Corner Radius = 36 px, Fill = #FFFFFF. Hover the mouse by the corner of your shape to reveal a circle, the radius can be changed there.
- Add Drop Shadow: X=0, Y=4, Blur=16, Opacity=25%, Color=#000.
- Name: Calculator Body.



On the left sidebar move the shape to become a child of iPhone 16 Calculator Frame. This will automatically center it.



Add Solar Panel

Press R to draw a rounded rectangle at the top of the calculator body.

- Width: 160 px Height: 28 px
- Corner Radius: 4 px (zoom in and select)
- Fill: #4A4A4A
- Name: Solar Panel

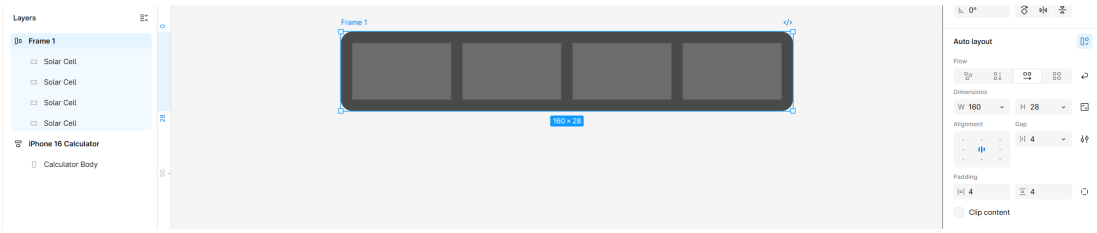
Inside this main rectangle, draw four smaller rectangles (the “solar cells”).

- Each cell: 35 × 20 px Fill: #6C6C6C

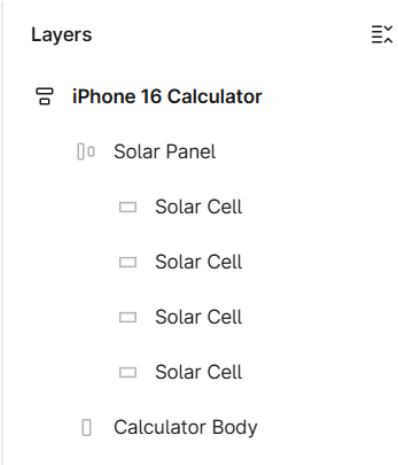
In the Layers panel, drag the four cell rectangles into the main Solar Panel (right click, bring to front if needed)

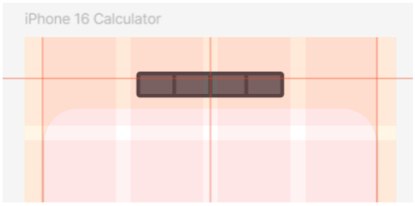
Select the Solar Panel parent layer and press Shift + A to apply Auto Layout → Horizontal.

In the right sidebar, set:



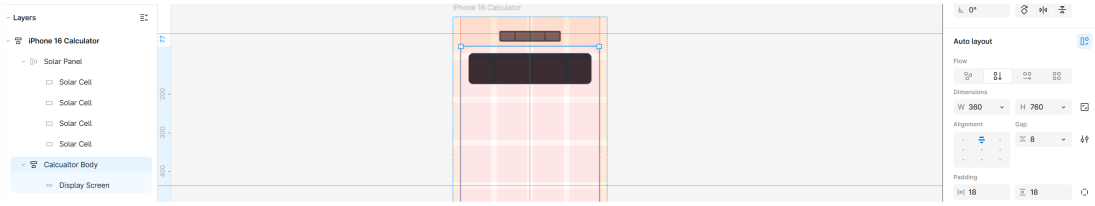
Rename the group to Solar Panel and drag it to become a child of the iPhone 16 frame just above the calculator body, it will automatically be placed on the frame and positioned.





Display Screen

- Rectangle 320 × 80 px, corner radius 12 px, fill #263238.
- Make a group for the Calculator Body by selecting the calculator body and pressing control G or right click and press group, then move the Display Screen on the left panel to become a child of the Calculator Body.
- 4. Select Calculator Body and press Alt + Shift + A for auto layout and choose alignment as top center, set the Gap to 8.



Add text 'four plus five equals nine' (Poppins 20 px, color #FFE600, semi-bold). Select the screen and press control G to make it a group and name it Display Screen, put the text inside the screen and center both over the center red line, drag the corners of the text box to the corners of the screen. Then click the display screen group and enable Auto Layout using Control + Shift + A. Set Flow to vertical. Set the alignment of the text box to center and middle.

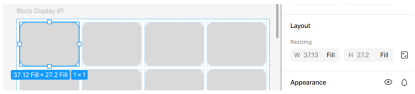
Counting Blocks – Visual Math Area

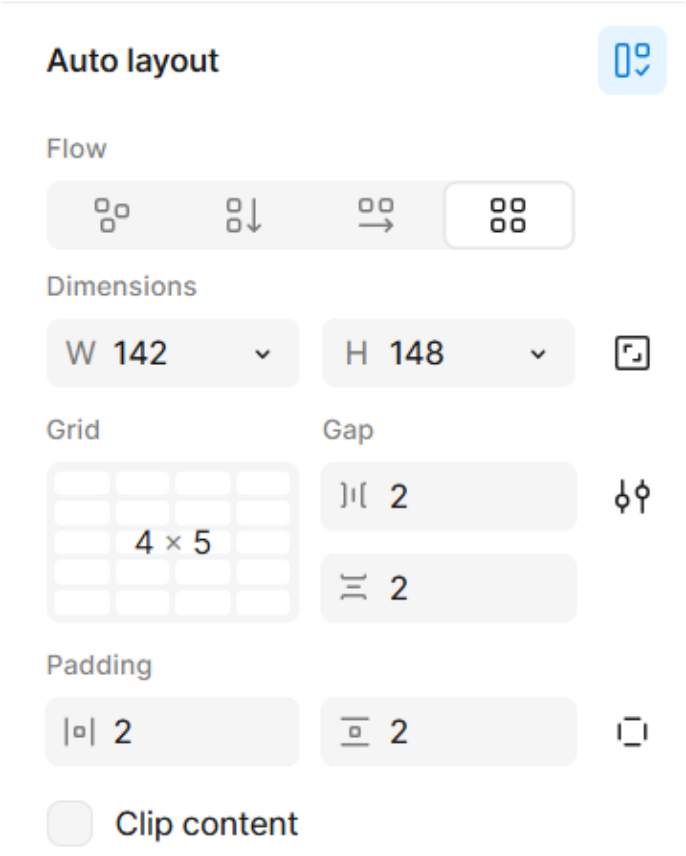
Please look ahead and if you want to exclude this feature you may do so and retain many keys features of the calculator. However this step provides excellent practice in creating dynamic groups that can dynamically change in shape.

- Make a square with width 140 and height 140, set the color to white.
- Add a small square in the top left corner of the larger square width 33 and height 27
- Set the corner radius of the small square and the large rectangle to 4
- Copy and paste the squares across the row to make a row of 4 squares and then copy them down to have 20 small squares total inside.
- Select them all and press control G to make a group then press shift A to enable auto layout. The flow should

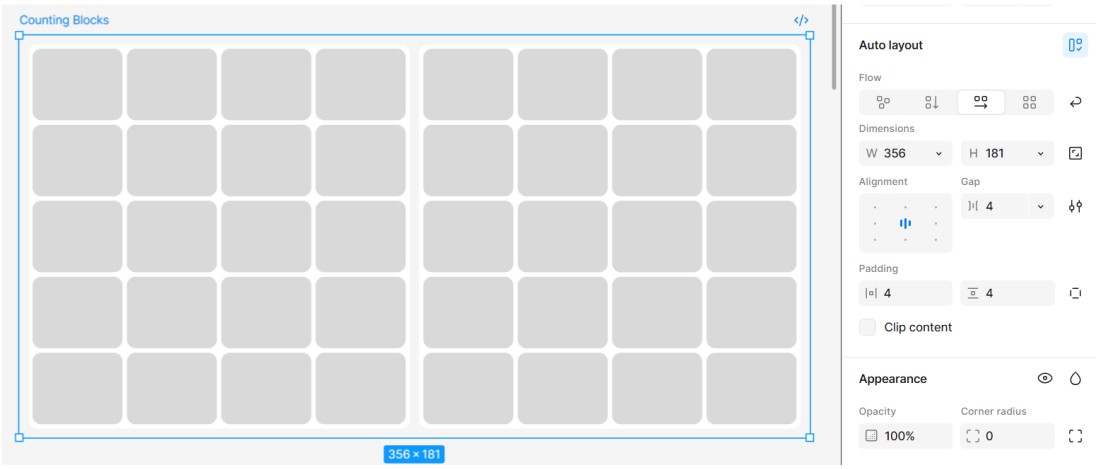
be set as Grid, set the Gaps and the Paddings to 2 each. Rename the group to Block Display.

- After shapes are grouped, the children shapes can have their width and height set to fill which allows them to change size according to the parent. Select the children shapes and on the right panel set resizing to fill for height and width.

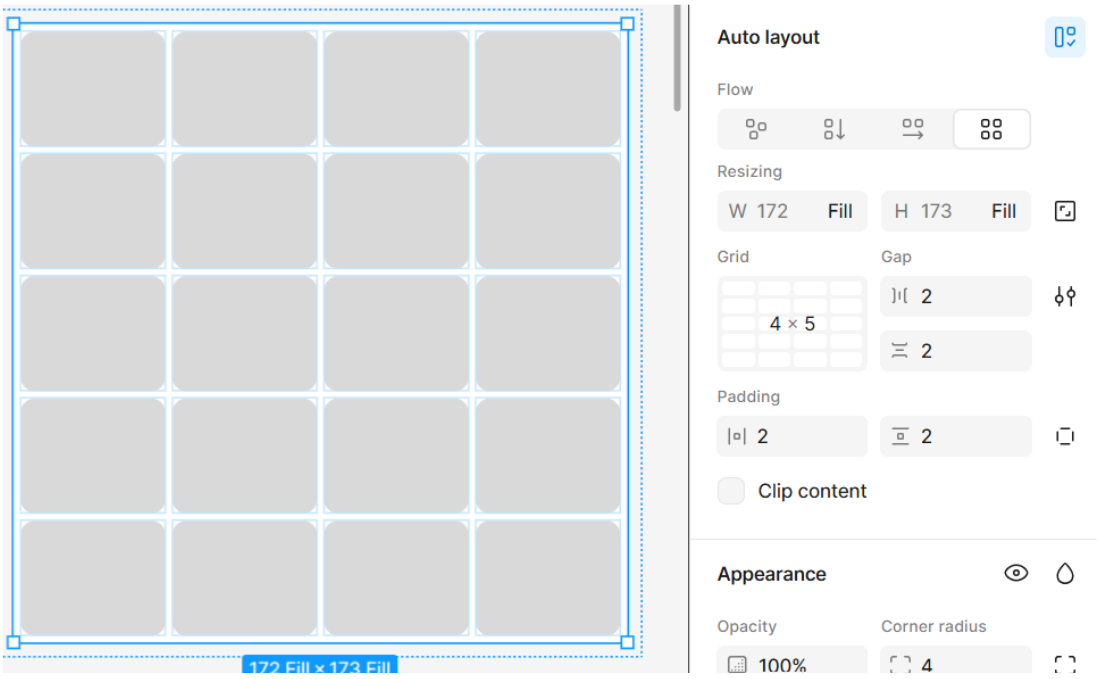




Copy the block display and place the 2nd one to the right, name is block display 2. Select block display 1 and 2 and make them a group control G. Set the layout to auto horizontal with gaps and paddings of 4 and alignment center. Name the frame counting blocks.

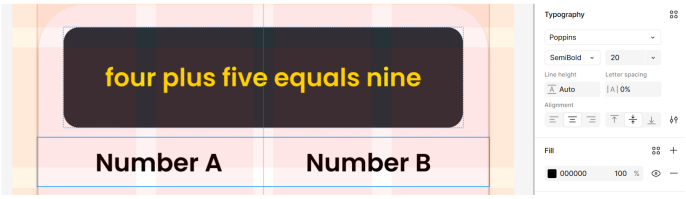


Select the 2 children block displays and set their sizing to fill for both width and height.

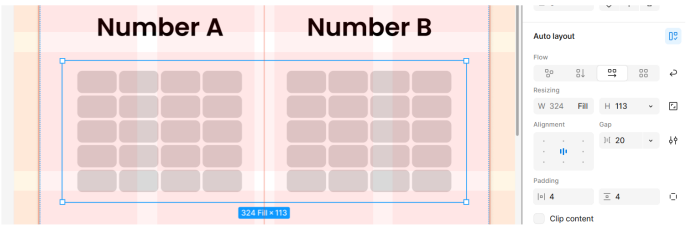


Follow these same methods to create a group called total blocks with a total number of 40 blocks, it should be its own independent group and the squares inside should adjust height and width dynamically.

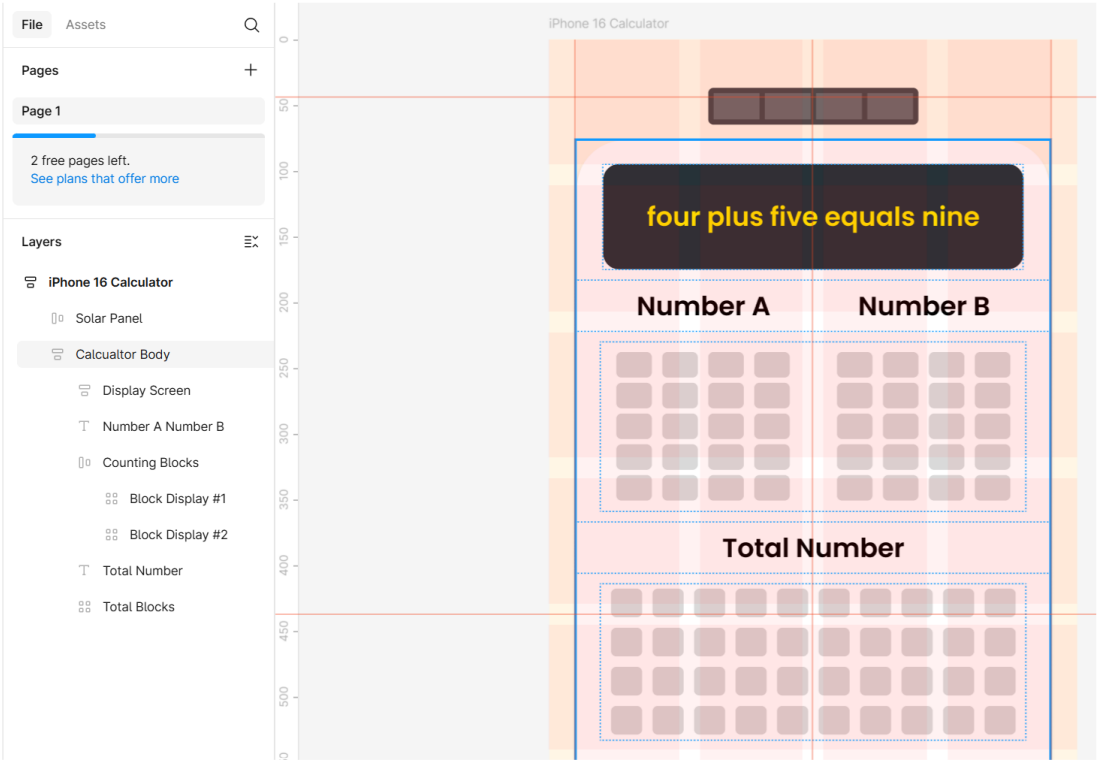
Place a textbox on the phone under the display screen and add the text. Number A (14 spaces) Number B. Center align it.



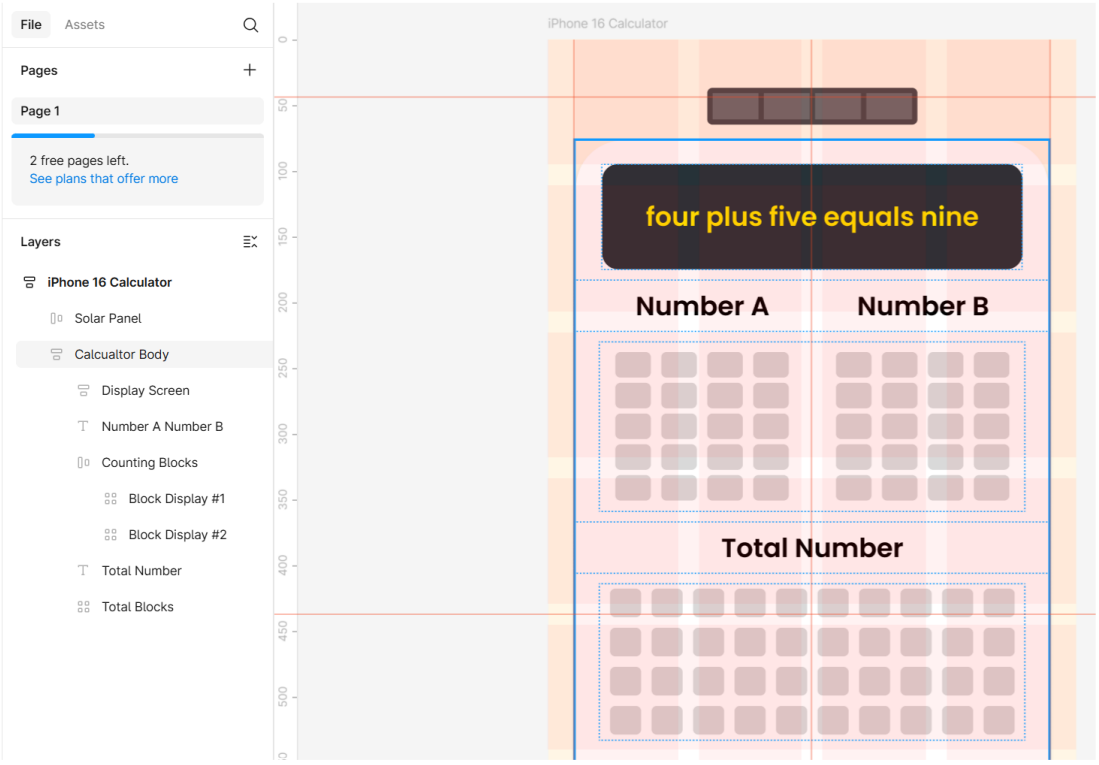
Add your Counting Blocks group to the phone and make adjustments, I reset the frame size to width 324 and height 113 with a gap of 20.



Copy the Number A Number B text box and add it below, change it to Total Number. Then add the Total blocks group below.

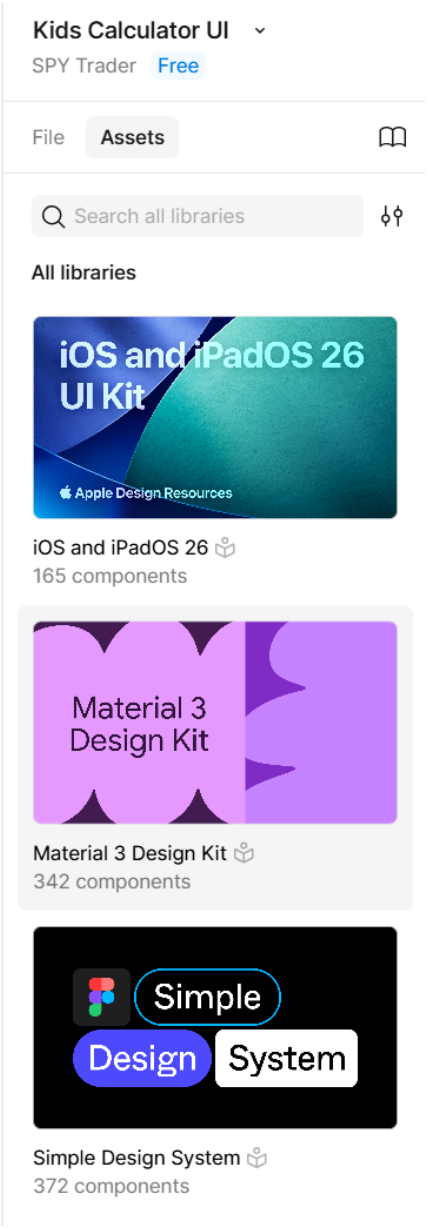


I made slight adjustments to position and padding of Block Display #1 and #2 by selecting them both together from the left side menu as well as adjustments to my total number to get the desired look I was looking for which may be adjusted more later.



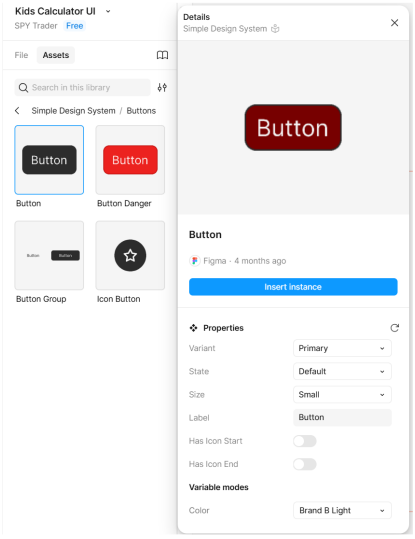
Buttons Grid Layout

On the left panel click on Assets at the top to access the Asset library which includes many reusable parts excellent for user interface designs.



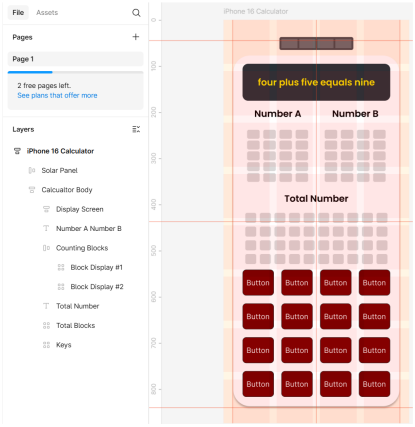
Open the Simple Design System by Figma.

Select the regular button, change the color variant to Brand B Light. Insert instance.



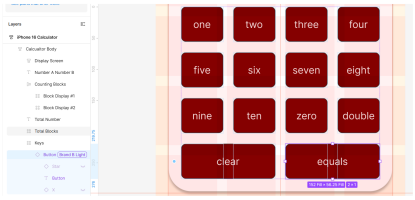
Make a row of 3 buttons across and columns of 4 buttons down. Group the selection with control G.

Toggle auto layout grid with shift A. name it kays and p[lace it under total blocks using the left side menu to position it.

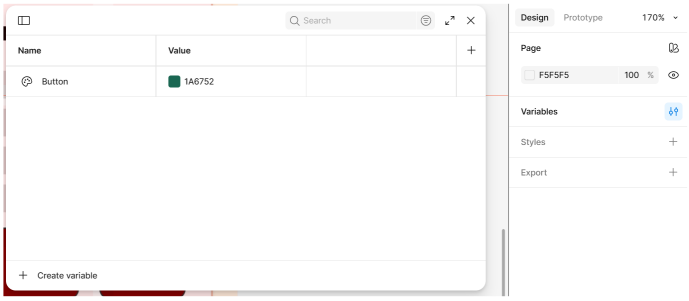


Edit the buttons

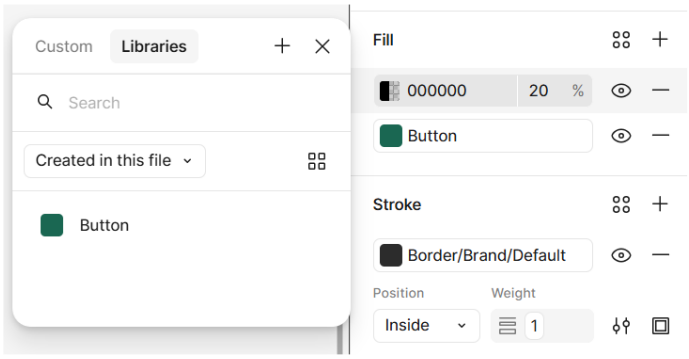
Click the buttons to change the text, label them one to ten using words then a key for zero and double as well as clear and equals. At the bottom remove two extra keys and stretch the clear and equals keys across to fill the empty space.



Edit the font style and button colors. With nothing selected at the top left you can find a menu option that says variables, click it and enter a color there and name it Button.



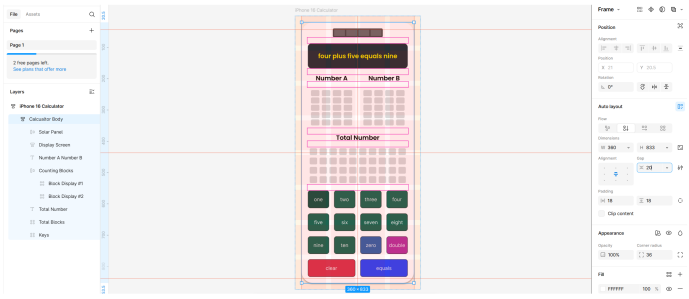
Go to your number buttons and change 1-10 to this color. Click the number button, remove the old color fill, add a new fill, select libraries and insert your color. Press shift, select and click the rest to change them all.



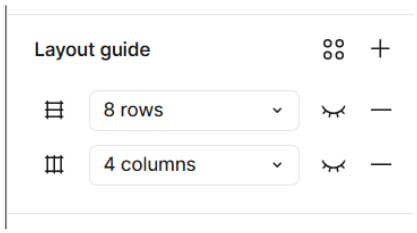
Select unique colors for the other buttons individually.

Adjust Master Layout

- Move the solar panel inside the Calculator Body, nest it at the top.
- Stretch the body a little more vertically for top and bottom.
- Set alignment to center and increase gap to 20.

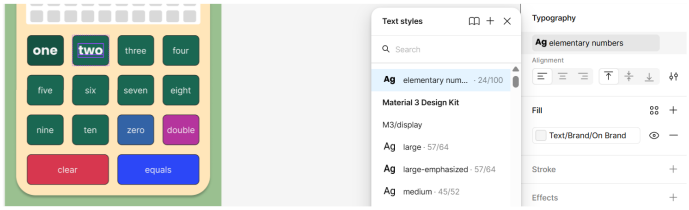


Turn off your layout grid and move the guidelines back into the ruler.

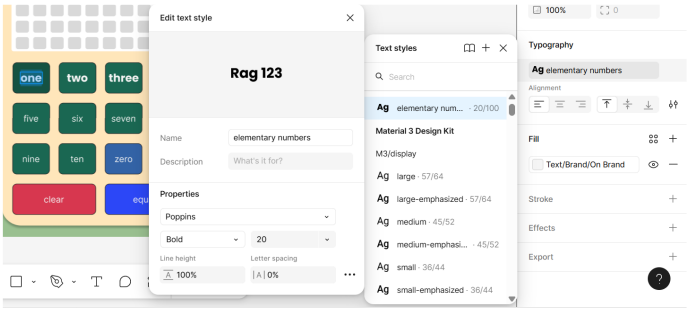


Set alignment to center and increase gap to 20.

Adjust the typography of your number buttons. Remove the current typography and create a new style using the mini menu options.

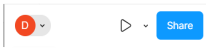


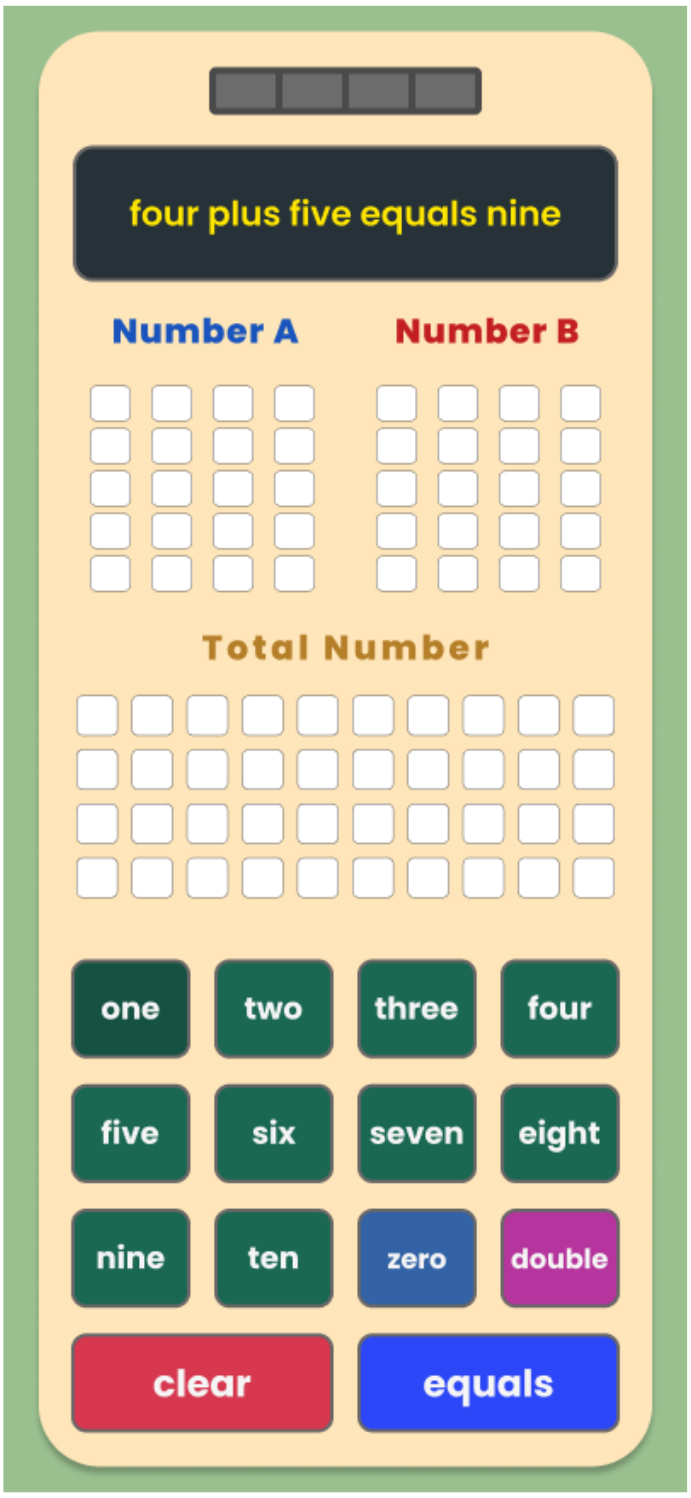
For the other buttons you can click the typography, then scroll to the top and select the one you created. When you later edit this style, all the text with the same style will update.



Keep adjusting you styles and components until you reach a desired outcome. I added an invisible square to use as a spacer in my design because the keyboard was too close to the total number blocks. Have a look at my final design below.

Press the play or present in the top right corner to view how the design would look on a phone.

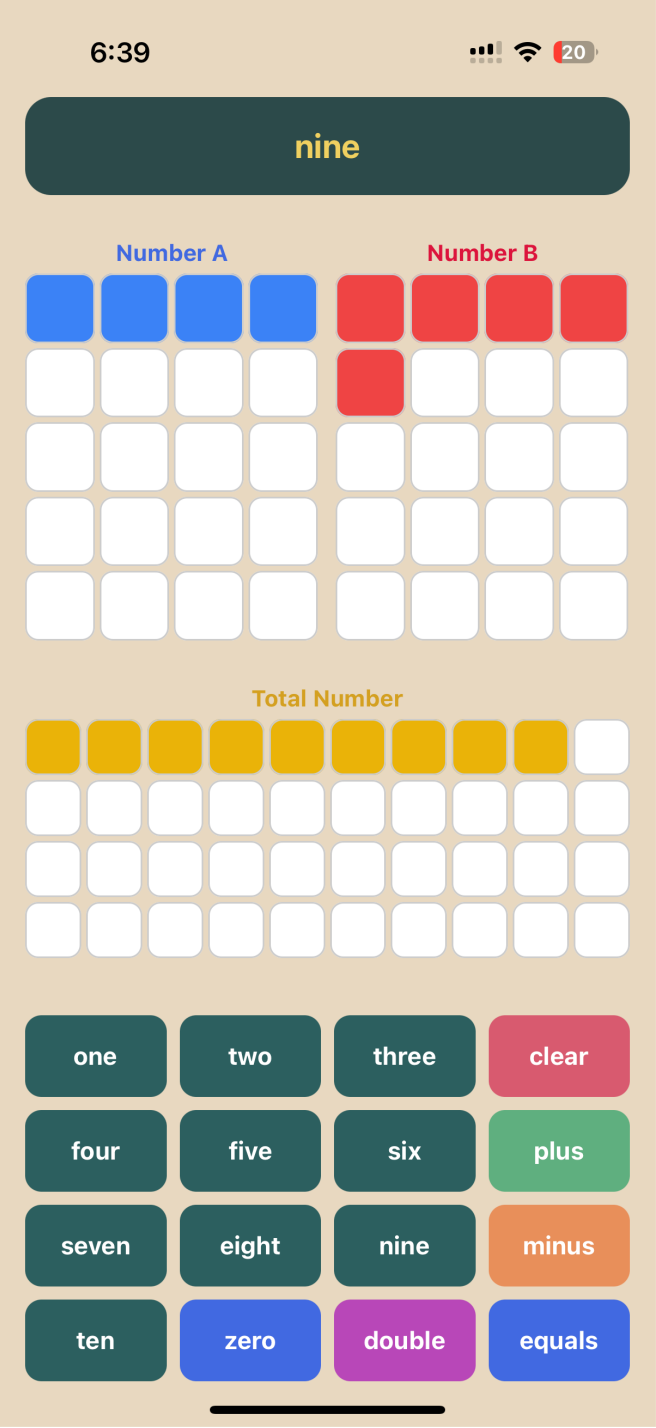




Export Your Design

- Select frame 'iPhone 16 Calculator'.
- Right-click → Export → PNG (2×).
- Save as KidsCalculator_UI_Final.png.

After you begin developing your app, you'll make various modifications. Below is an update to show some of those modifications I made. This is a snapshot from the actual app running on my iPhone.



A completed interface design should communicate layout, spacing, and visual hierarchy clearly enough that it can be implemented consistently in code. Before exporting, verify that your design aligns to the grid, uses a safe area margin, and keeps repeated elements such as buttons and counting blocks consistent in size and spacing. Use clear layer names and reusable text styles so the design remains easy to revise when requirements change. Exporting a final frame as a PNG provides a stable visual reference for development, and it also creates a baseline for future iterations when you compare the design to the running app.

What Figma is used for and why it is popular

Figma is a collaborative design platform used to create interface layouts, interactive prototypes, and shared design systems for web and mobile software. It is commonly used to design screens, define reusable components, standardize typography and color styles, and collect feedback in a single shared file. Figma is widely adopted because it supports real-time collaboration in the browser, which makes it accessible across devices and simplifies sharing design work with classmates, teachers, and development teams.

Chapter 8 Design a User Interface on Figma Rubric

This chapter focused on designing a mobile app interface in Figma, using layout tools, organizing visual elements, improving usability, exporting the design, and documenting the design process.

Category	Points	Evidence of Completion
Project Setup	5	A Figma design file is created, clearly named, saved, and organized with an appropriate mobile frame or screen size.
Interface Layout	5	The app interface includes a clear structure, organized sections, readable spacing, buttons, labels, and required screen elements.
Visual Design	5	Colors, fonts, shapes, icons, and visual hierarchy are used effectively to create a clean and appropriate app design.
Usability and Revision	5	The design is reviewed for readability, spacing, alignment, button size, and ease of use. At least one design issue or improvement is revised.
Evidence of Work	5	Screenshots, exported images, Figma file links, design layers, or other project evidence are submitted.
Reflection	5	A short explanation describes the app design, the design choices, the revision process, and relevant computer science or design vocabulary such as user interface, frame, layout, alignment, component, prototype, accessibility, spacing, or visual hierarchy.
Total	**30**	

Submission Checklist

Requirement	Check
Figma design file created	☐
Project clearly named and saved	☐
Mobile frame or screen size selected	☐
Main app interface completed	☐
Buttons, labels, and screen elements included	☐
Spacing and alignment checked	☐
At least one design issue or improvement revised	☐
Final design exported or screenshotted	☐
Figma link or project proof included	☐
Short reflection completed	☐

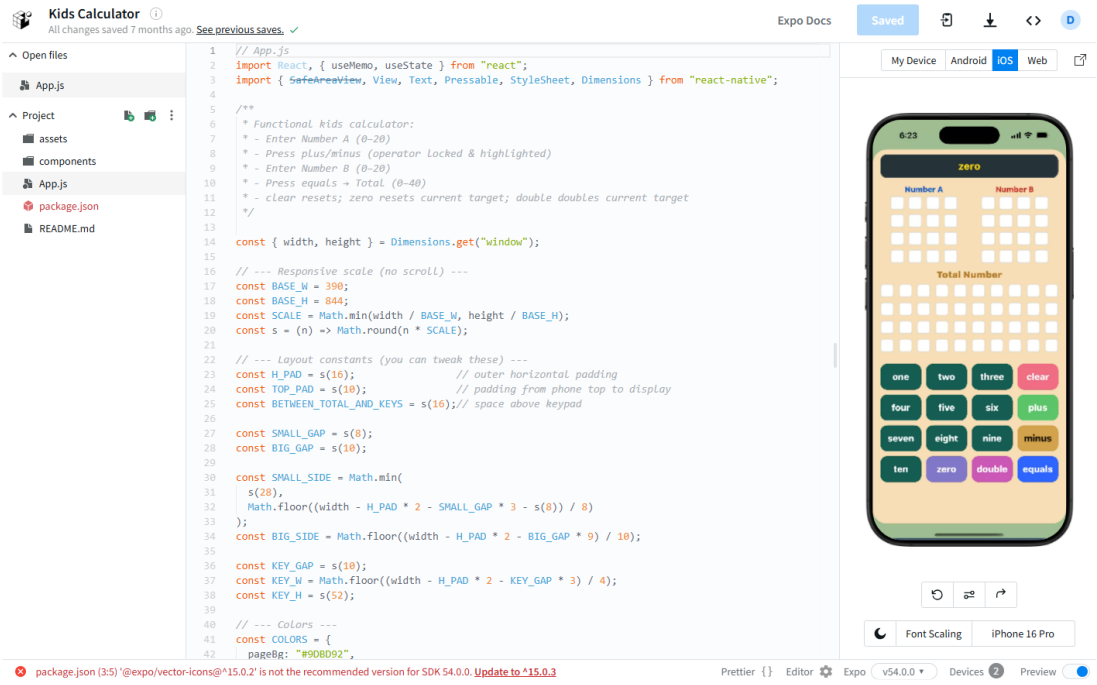
Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the app interface created in Figma. Explain one important design choice related to layout, color, spacing, or usability. Identify one part of the design that was reviewed and improved before final submission.*

9

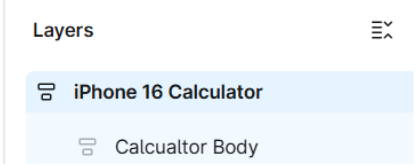
Convert Figma Design to Code

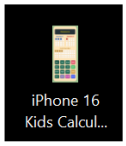


This lesson demonstrates a common workflow for turning a visual mockup into a working prototype by converting a Figma screen into React Native user interface code. You will export your main frame as a high-resolution PNG, use an AI agent to generate Expo-compatible React Native layout code, and paste the result into Expo Snack, which runs React Native projects in the browser. You will then preview the same interface on iOS, Android, and a physical phone using Expo Go and a QR code, and iterate until the layout fits the screen and matches the original design.

In Figma

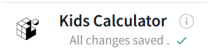
Select the main frame, on the right panel scroll to the bottom and export the image as PNG. Choose 2x to make it more detailed. Name it a name plus UI for User Interface.



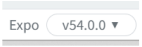


Install Expo Go on your phone

- Install Expo Go on your phone so you can scan QR codes and try your app.
- Convert Image to Code using an AI agent
- Open Snack and set the SDK to the newest version
- Open Snack at https://snack.expo.dev/ and create or rename your project.



Set the SDK to the newest version at the bottom-right corner.



Upload your PNG and paste the provided prompt

Open your AI chat. Upload your PNG image and paste the provided prompt.

Prompt:

```
I uploaded a PNG of a mobile app UI. Please generate React Native code for Expo
that I can paste into https://snack.expo.dev/. Follow the design I uploaded
closely, only create code for the user interface, later we will add the code
for functionality.
```

Copy the AI's generated code and paste it into Snack → App.js.

Preview your app on iOS, Android, and My Device

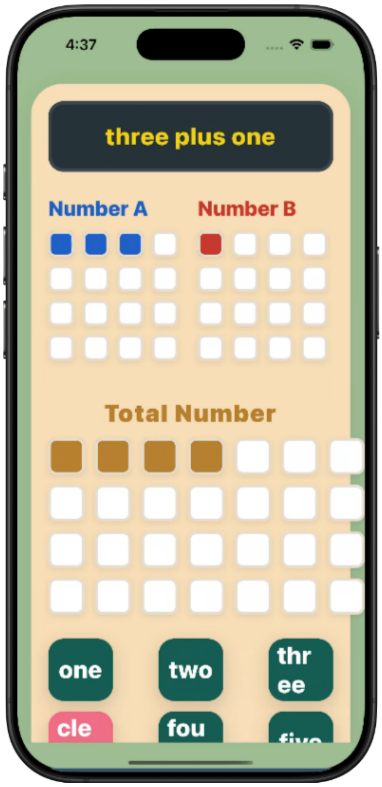
On the right panel choose iOS, Android, and My Device. Press Launch Snack or scan the QR code with your phone.

Refine your UI and regenerate code if needed

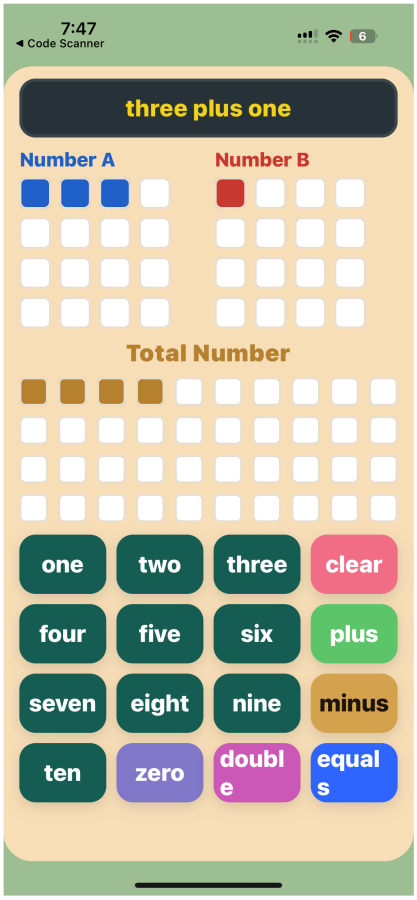
These apps are often not well designed for web view but if you use web view you can adjust the window size to check the responsiveness of the app.

If errors occur, re-ask the AI to fix and regenerate the code with a different prompt and try again.

Take a screenshot of your app and paste it back in your AI chat and make some adjustment requests. Can you update the code? I would like everything to fit on the screen without scrolling. Attached is a screenshot of what the given code produced.

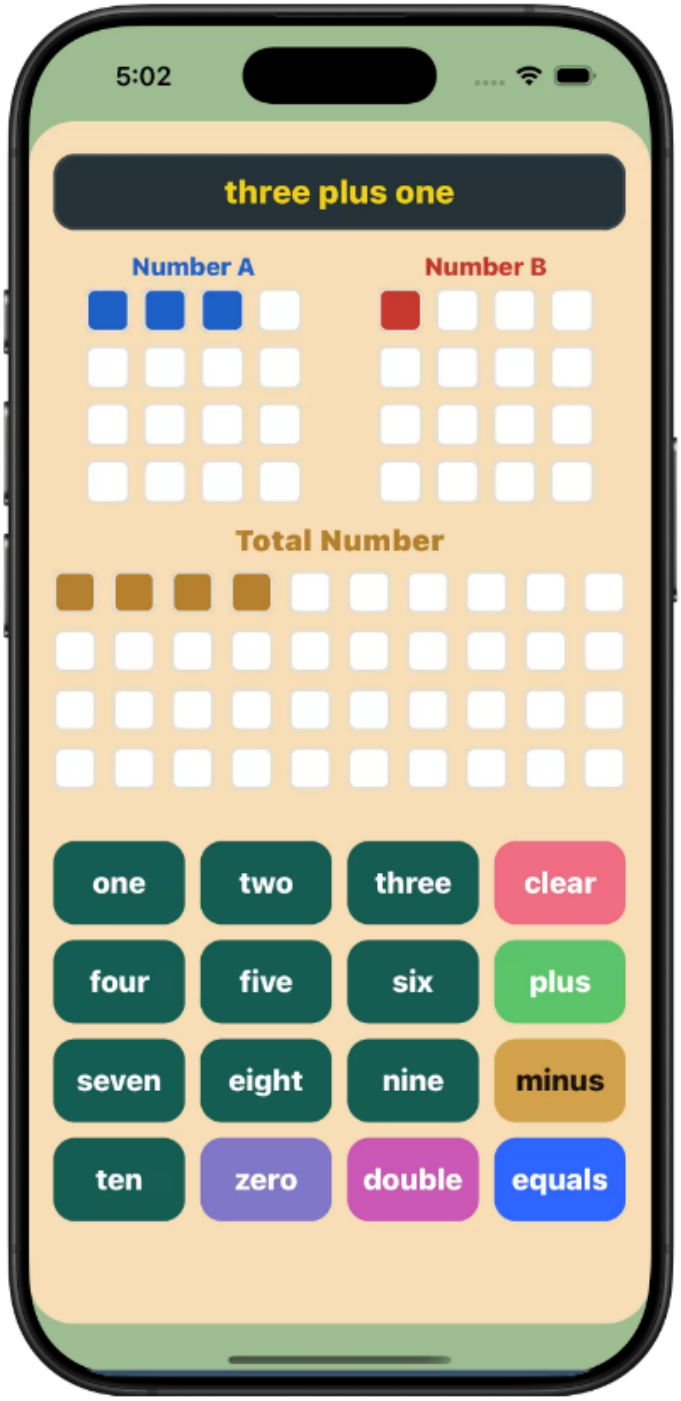


Upload the new code to snack.expo.dev, likely you'll see some improvements.



To make further refinements you could write a prompt to your AI agent like, can you slightly reduce the size of the text? Can you center the Number A and Number B text and boxes? Can you add more padding between the total number boxes and the keypad? You can try these or ask your AI agent, where can I find the code to change the size of the text? Please provide the line number. etc. Likely you will need to do this to fine tune your design anyhow. Inside snack.expo.dev you may use control f to search for keywords in the code too.

Fully refined with one prompt and 2 manual adjustments



Start adding functionality to your code

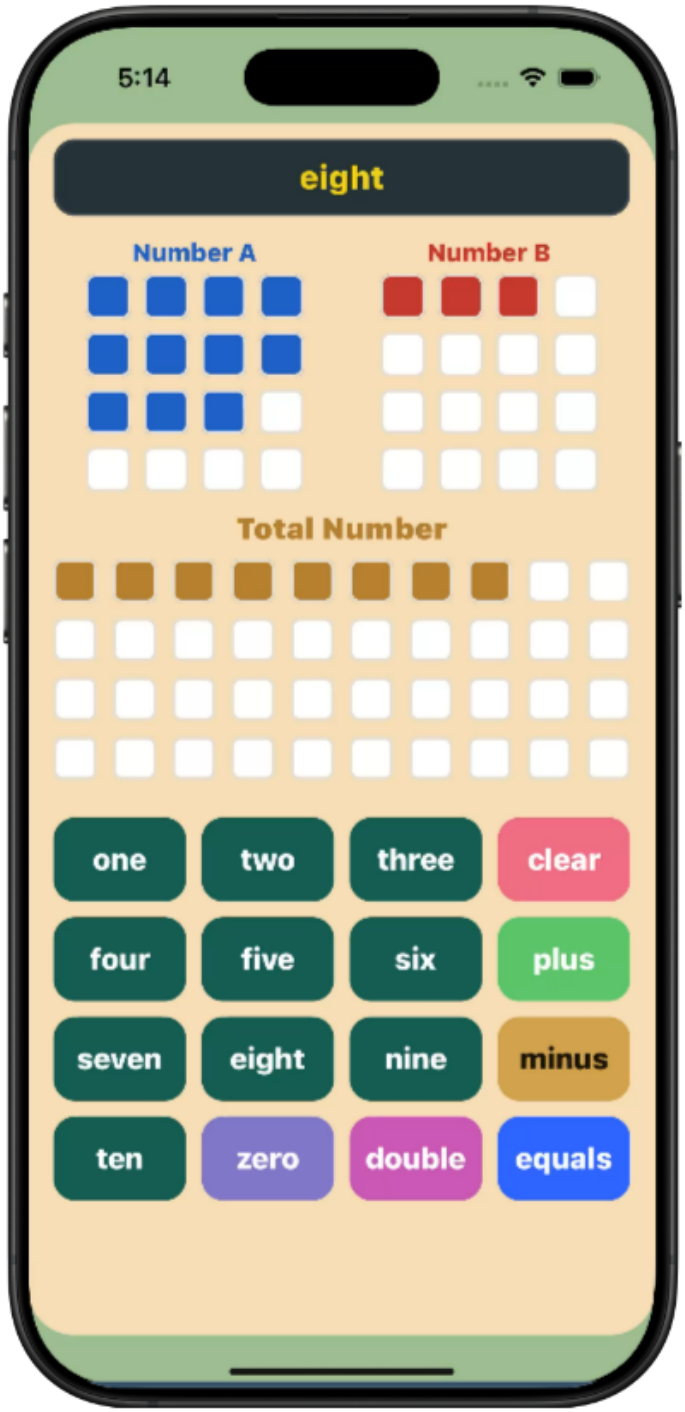
After you have made all your adjustments, you may start adding functionality to your code. Some code requires multiple files, consult your AI agent about what suggestions it has in order to make the app fully functional. You may also upload your most up to date picture.

Prompt:

```
Here's my updated user interface. I would like to make this app fully functional.
Should we keep everything in one file or use multiple files? Either way is fine.
Do you need any additional information from me to make this app fully functional?
This is a calculator for kids. The first numbers pressed go into Number A and
the boxes light up according to the number with a limit of (20). After the user
presses plus or minus, the input goes to Number B (up to 20). After the user
presses equals, the total number is displayed (up to 40)
```

Download the code as a ZIP file and save it for future use

If your prompt worked then you have a fully functional app. You may keep ironing out the issues until you reach the desired result. Make sure to report to your AI agent the exact behavior and give clear example of what your goals are. Here's mine below ready for export.



Note: If you want to add sound effects you can ask the AI agent to update the code to include sound effects using expo-av, for spoken content use expo-speech, for the full list of Expo SDK capabilities visit: https://docs.expo.dev/versions/latest/sdk/expo/

From here you should download the code as a zip file and save it for future use or reference. Even if you don't publish the app, it may have features in the code which can be helpful to you or provided to AI agents to assist in remaking those components.

Next Steps

Now that you have successfully used your own design to build a working coded project you may want to consider moving the files over or trying the same steps using Bolt. New, Cursor, Visual Studio Code, or Xcode. Just ensure that in the beginning you use the same setup by using the react plus expo framework. For Xcode you will be using Swift. Remember we picked react plus expo to make apps compatible for both Android and iOS devices, if your are designing computer apps or websites you may pick another framework and protype using other methods.

User interface code and why design often comes first

User interfaces are implemented in code by describing a hierarchy of visual components and layout rules. In React Native, the screen is built from nested components such as views, text, and buttons, and the final appearance is determined by properties such as size constraints, spacing, alignment, and styling. A visual design is often created before the coded interface because it establishes the layout requirements early, helps identify the information hierarchy, and provides a stable reference for consistency across screens and devices. This reduces rework by making decisions about spacing, typography, and component behavior before time is spent implementing logic.

Visual designs also support reuse inside a code project when they are treated as a component library rather than a single image. Repeated elements in Figma, such as buttons, blocks, labels, and display panels, map naturally to reusable code components, and shared design choices such as colors, spacing, and text styles can be translated into shared style constants or theme files. This is the basis of a design system: the same UI elements are defined once and reused throughout the app, so updates remain consistent and faster to apply.

Chapter 9 Use a Figma Design in an App Rubric

This chapter focused on converting a Figma design or exported image into app code, testing the interface in Expo, improving accuracy, adding basic functionality, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	A Figma design, exported image, or design screenshot is prepared and placed into an app development workspace such as Snack Expo, VS Code, Cursor, or another approved tool.
Interface Conversion	5	The app interface is recreated in code with matching layout, colors, text, buttons, spacing, and major visual elements from the original design.
App Functionality	5	The coded interface includes working interaction, such as buttons, input areas, displayed results, navigation, sound, speech, or another approved app feature.
Testing and Revision	5	The app is tested in preview or on a device. Differences between the design and coded version, layout issues, or broken interactions are identified and revised.
Evidence of Work	5	Screenshots of the original design and coded app, exported design files, project links, code files, or other project evidence are submitted.
Reflection	5	A short explanation describes how the design was converted into code, what was tested or revised, and relevant computer science or design vocabulary such as user interface, component, layout, image export, React Native, Expo, state, event, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
Figma design, image export, or screenshot prepared	☐
App project created or opened	☐
Interface recreated in code	☐
Major layout, colors, and text matched to the design	☐
Buttons or interactive features completed	☐
App tested in preview or on a device	☐
At least one layout issue, design mismatch, or functionality issue revised	☐
Original design evidence included	☐
Coded app screenshot or project proof included	☐
Short reflection completed	☐

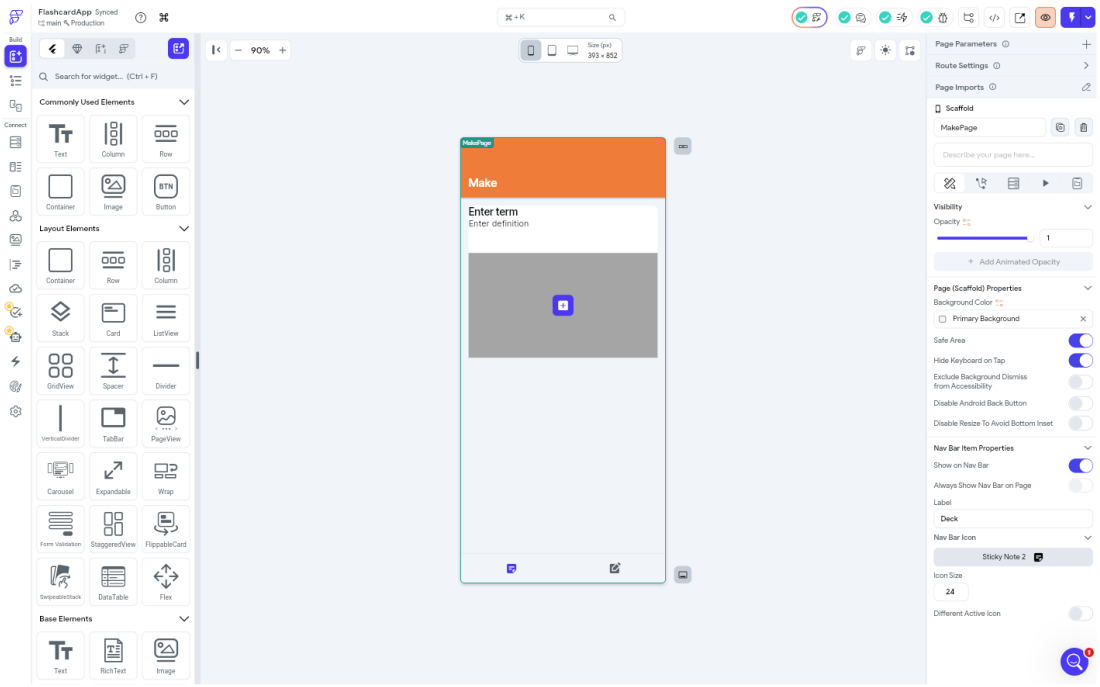
Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe how the Figma design or image was converted into app code. Explain one part of the coded app that matches the original design. Identify one layout issue, design mismatch, or functionality problem that was tested and improved before final submission.*

10

Make User Interface using FlutterFlow



This lesson introduces FlutterFlow as a visual development environment for building a simple flashcard app interface with two screens: a Deck page for viewing cards and a Make page for entering a term, definition, and image. You will create a new project, rename the default page, add a second page, enable a navigation bar, and build each screen by assembling common interface widgets such as columns, list views, containers, text, images, and text fields. You will then use the View Code feature to inspect the generated Flutter code and prepare the project for future functionality and export.

What You'll Make

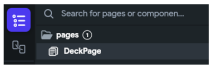
A flashcard app that allows the user to upload a term, definition, and picture to study with. The app will have two pages, one for Deck and one for Make.

Create Project

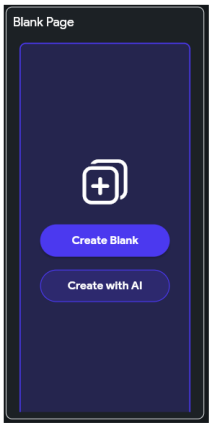
- Go to flutterflow.io and create an account, don't sign up for a plan.
- Click Create New, name it FlashcardApp, click Start Building.

Create Your Pages

- Open Widget Tree or press Tab 2
- Rename HomePage to DeckPage by double clicking it.

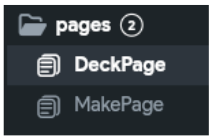


Add a new blank page by pressing the green plus button

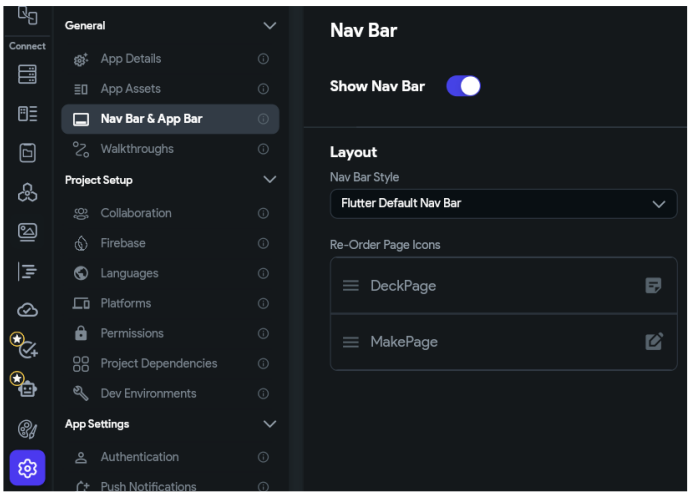


Name it: MakePage

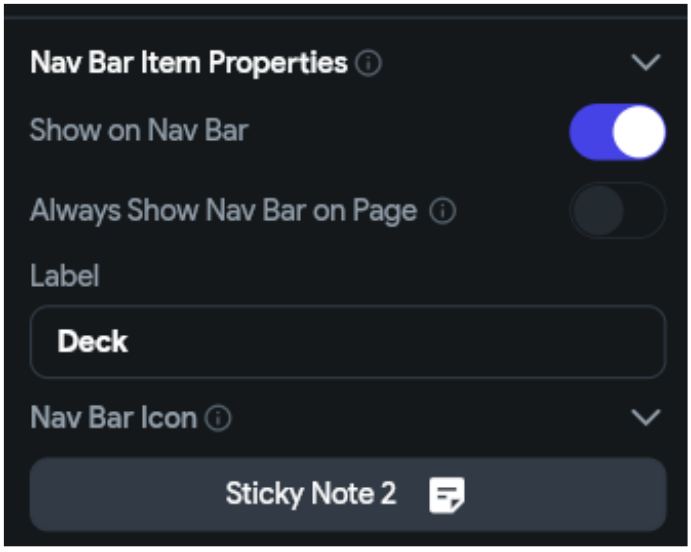
Click and drag it into the same folder

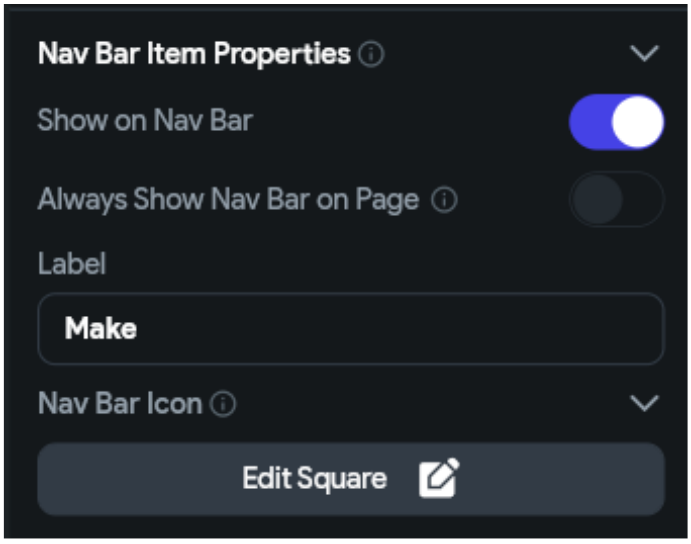


Inside settings turn on: Show Nav Bar



For both Deckpage and Makepage find the Nav Bar Item Properties on the bottom right hand corner and enable show on Nav Bar. I changed their icons to Sticky Note 2 and Edit Square.

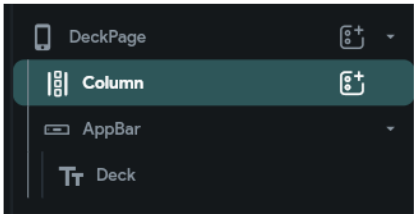




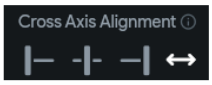
Make the Deck Page

Change the AppBar title to My Deck, by clicking the text box and changing the value on the right side.

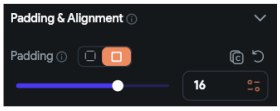
Click on the Column on the left side menu.



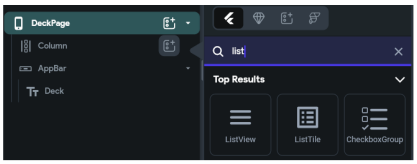
Change the Cross Axis Alignment to Stretch

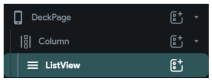


Under Padding and Alignment enable uniform padding and set the value to 16.

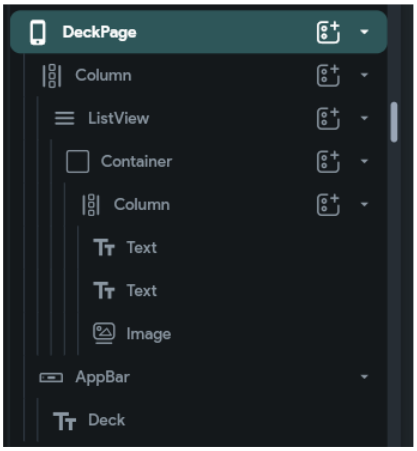


Add a child widget to the column, by pressing the button next to the column, search for list and select list view.

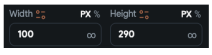




Add a container as a child of ListView. Add a column as a child of Container and add 3 children for the Column, Text, Text, and Image.

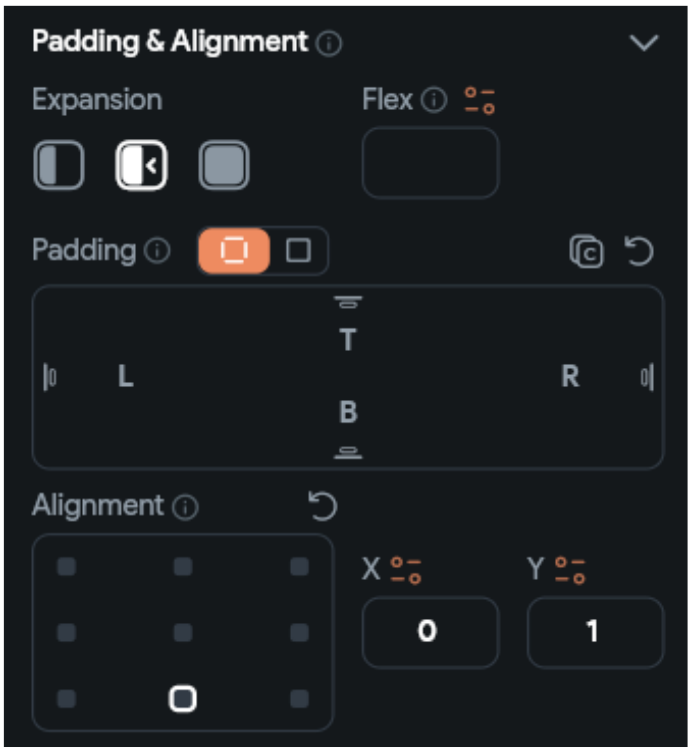


Change the container height to 290

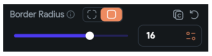


Change the Column's Cross Axis Alignment to Start.

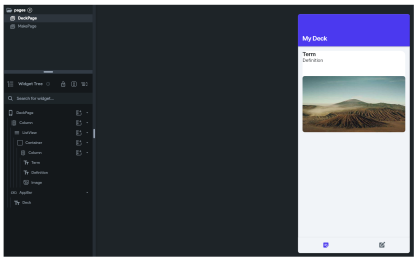
Choose the image, change expansion to Flexible, and alignment to bottom middle.



You may set the border radius of your Container to 16

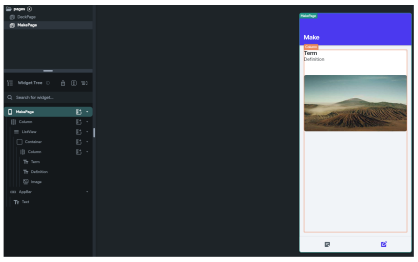


Change the Theme Text Style of your Term to Title Large. Set the definition font to body large.

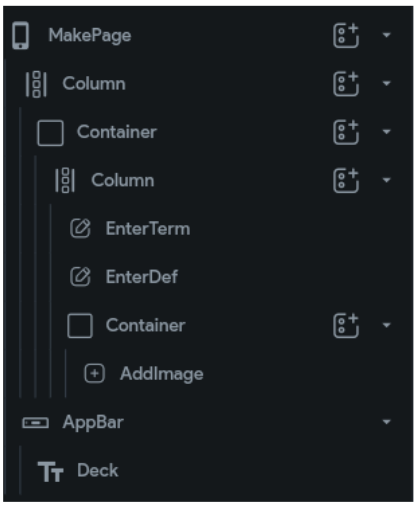


Make Page B

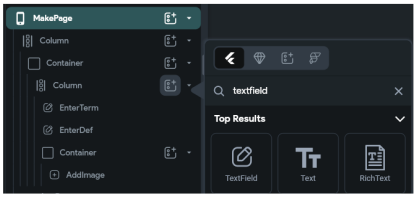
Go to Make Page and delete the column. Then go to your Deck Page and copy the column. Paste that column inside the Make Page and move it to the same position where the original column was before. Rename the App Bar to Make, change its color to orange.

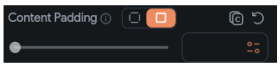


We aren't going to use the ListView component so drag and drop the column below it above list view quickly. If a dialog asks you to replace something hit cancel and try again. Then delete the ListView.



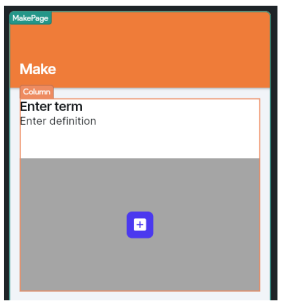
Delete the term and definition text boxes and replace them with Text Fields. Rename them EnterTerm and EnterDef. Set their font styles to Title Large and Body Large. Click Uniform Padding for each. For Width set it to infinite by clicking the symbol for both. Set a Max line of 1 for Term and 3 for Definition.



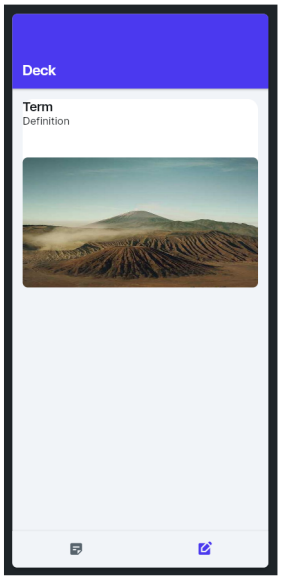


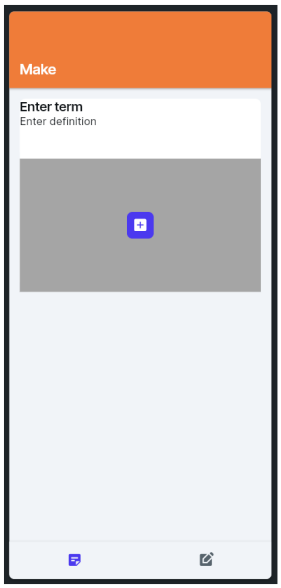
Delete the image and add a container in its place. Set the height to 200, color to light gray, alignment bottom center.

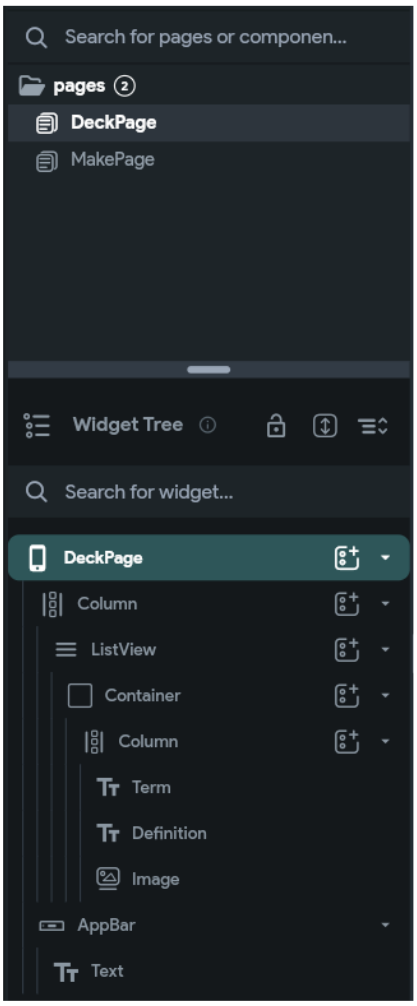
Inside the gray container add a widget called Icon Button, using the right panel at the bottom change the icon to add circle outline sharp.

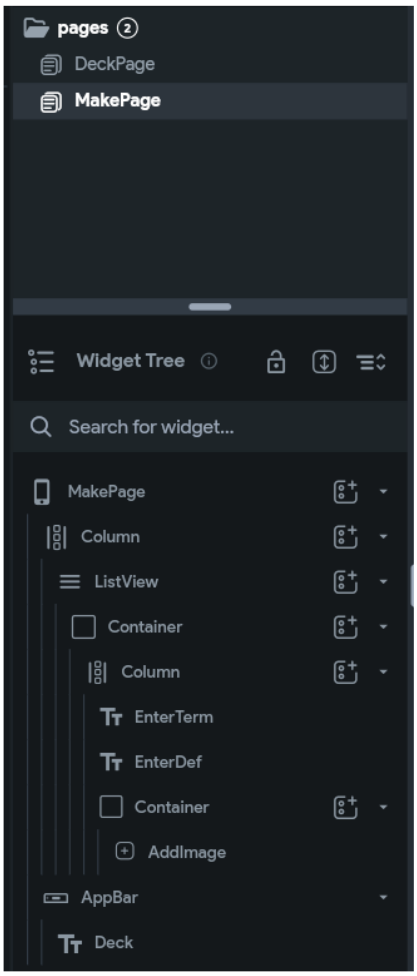


Now Let's take a look at both pages. Check your Widget Tree and modify any unnamed widgets with better descriptive names.



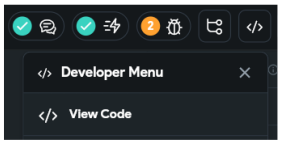



Search for pages or componen...
pages 2
DeckPage
MakePage
Widget Tree
Search for widget...
DeckPage
Column
ListView
Container
Column
Term
Definition
Image
AppBar
Text



Review and Revise Code

At the top right of your screen find developer button and click View Code.



You can paste the code for your Deck Page and Make Page into an AI chat with the prompt: Here's my Flutter Flow UI code, have a look and give me any suggestions or fixes.

At the end of this lesson you should have a complete user interface layout for a two-page flashcard app, with consistent spacing, clear widget names, and a working navigation bar between pages. The View Code step is an important checkpoint because it connects your visual design choices to the generated code structure you will use when the app becomes functional. At this point you can choose to export design screenshots, export code, or both, then translate the project into the development workflow you plan to use elsewhere, or begin connecting your interface components so the app functions inside FlutterFlow. One important consideration is that FlutterFlow includes a paywall near publishing through a monthly membership, so it is useful to decide early whether you will keep the project in FlutterFlow for release or move it into a local Flutter workflow after you complete your UI and initial functionality.

Flutter and FlutterFlow

Flutter is an open-source software development kit for building applications from a single codebase, and it uses the Dart programming language. FlutterFlow is a visual builder that generates Flutter user interface code while allowing the developer to assemble screens using a widget tree, properties panels, and reusable components. FlutterFlow is attractive as a development tool because it reduces the amount of manual layout coding needed to produce a working interface, supports rapid iteration through visual editing, and still allows a transition to a standard Flutter project when code export is enabled on the selected plan.

Chapter 10 Make a User Interface Using FlutterFlow Rubric

This chapter focused on creating an app interface in FlutterFlow, organizing pages and components, reviewing the generated design or code structure, testing the interface, revising problems, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	A FlutterFlow project is created, clearly named, saved, and organized with the required app pages.
Interface Design	5	The app interface includes organized pages, clear layout, readable text, buttons, images or placeholders, and required screen elements.
App Structure and Navigation	5	The app includes the required page structure, such as a deck page and make page, with clear organization and usable navigation or screen flow.
Testing and Revision	5	The interface is previewed or tested. Layout issues, missing elements, navigation problems, or design concerns are identified and revised.
Evidence of Work	5	Screenshots, FlutterFlow project proof, page previews, design layers, code view screenshots, or other project evidence are submitted.
Reflection	5	A short explanation describes the app interface, how FlutterFlow was used, what was tested or revised, and relevant computer science or design vocabulary such as page, component, widget, layout, navigation, user interface, prototype, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
FlutterFlow project created	☐
Project clearly named and saved	☐
Required pages created	☐
Main app interface completed	☐
Buttons, text, images, or placeholders included	☐
Page structure or navigation checked	☐
At least one layout issue, missing element, or design concern revised	☐
Project preview or page screenshots included	☐
FlutterFlow project proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

Describe the app interface created in FlutterFlow. Explain how pages, components, or widgets were used to organize the app. Identify one layout issue, missing element, or navigation problem that was tested and improved before final submission.

11
Make an App for iPhone on Mac using Storyboard

This lesson introduces iPhone app development using UIKit and Storyboard in Xcode. You will create a new Swift project that uses Interface Builder to design a calculator layout visually, including a display label and a keypad built with stack views and Auto Layout constraints. After designing the interface, you will connect storyboard elements to code using outlets and actions, then add Swift logic to make the calculator function. The lesson ends with testing in the iPhone Simulator and running the same app on a physical iPhone using Xcode signing and developer settings.

Install Xcode

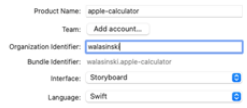
Open the App Store on your Mac and download Xcode.

- For older computers or operating systems you may need to download an older version of Xcode from apple developers website. https://developer.apple.com/download/all/?q=Xcode%2014.2
- I determined the version I would need by entering in my computer information and operating system into an AI chat and discovered I will need to use Xcode 14.2 downloaded from the developers website above. Find this by clicking the Apple icon in the top left corner and selecting: About This Mac

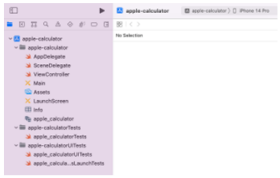
Create a New App Project

Open Xcode, Create a new Xcode project

Choose App (macOS → App) → click Next. Name it: apple-calculator. Set Interface to Storyboard. Set Language to Swift. Don't enable Use Core Data or Include Tests.



Save it to your Desktop or create a folder inside your drive, one for all projects and one inside that for your calculator project.

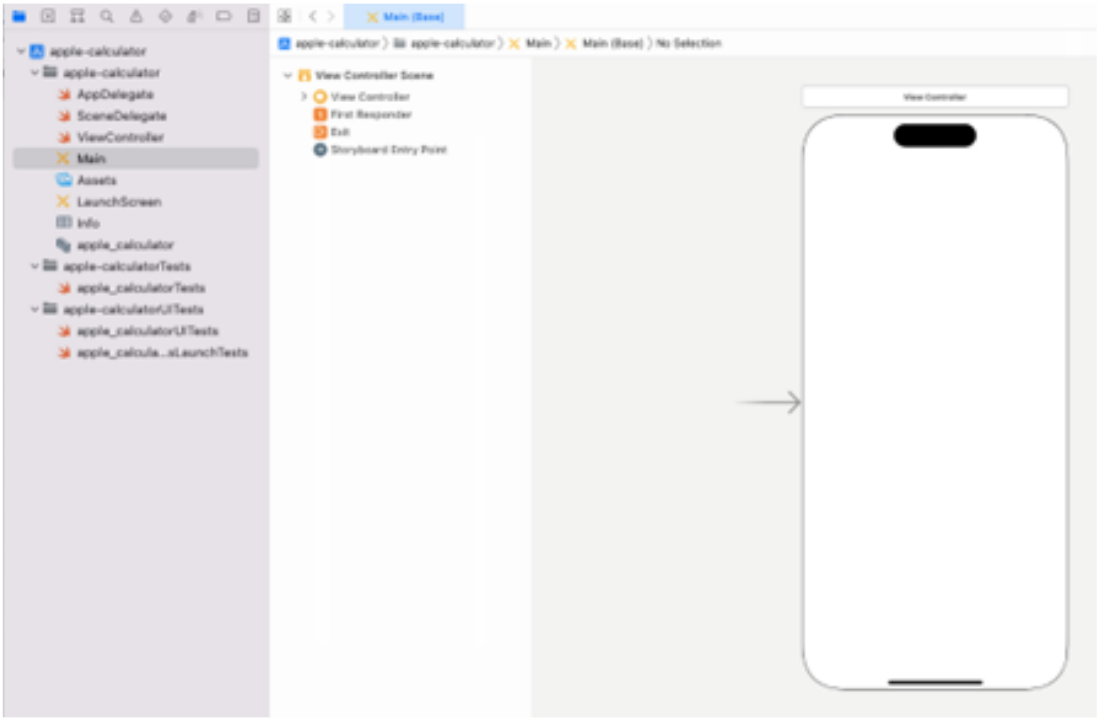


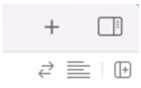
Here's the view inside my Xcode

Design the Calculator

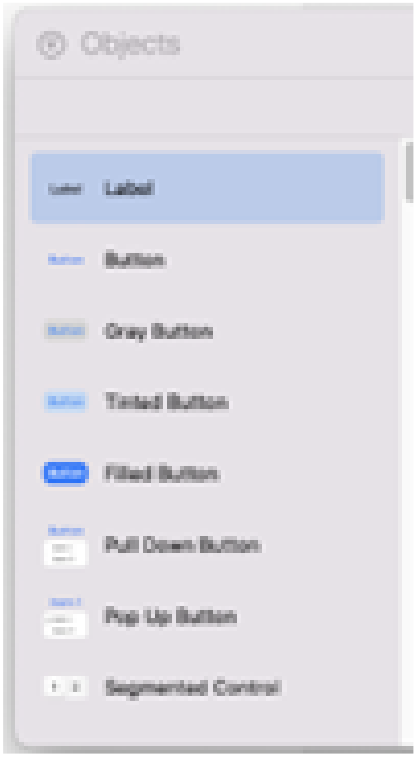
In the left panel, open Main.storyboard.

Drag and drop these items from the Library (+ button at the top right):

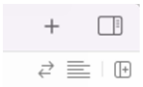




Add View under View Controller

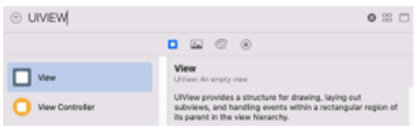


Here's what your file tree structure should look like

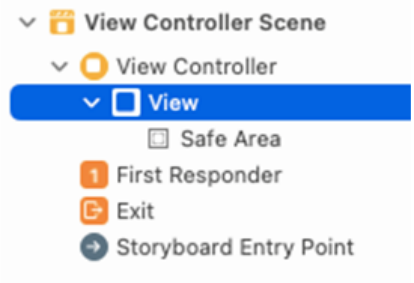


Add the display (Label) at the top

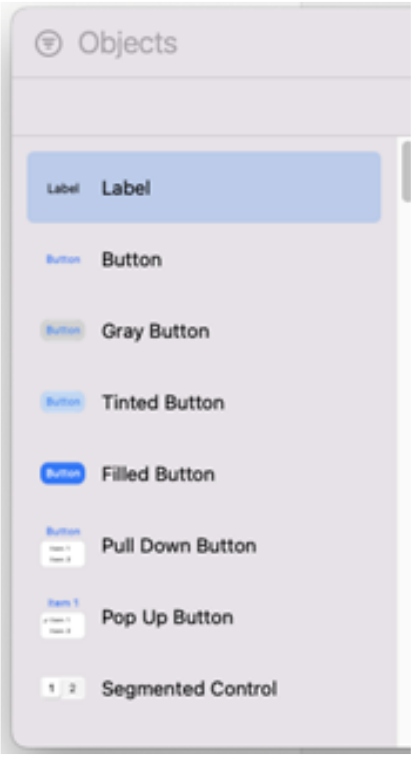
Object Library → drag a Label into the top area of the View Controller's view.



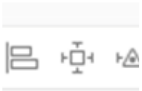
Add constraints (click the Pin button at bottom-right of Interface Builder):



- Check Top, Leading, Trailing → set each to 16.
- Check Height → set to 60 (we'll make it ≥ in a moment).
- Click Add Constraints.



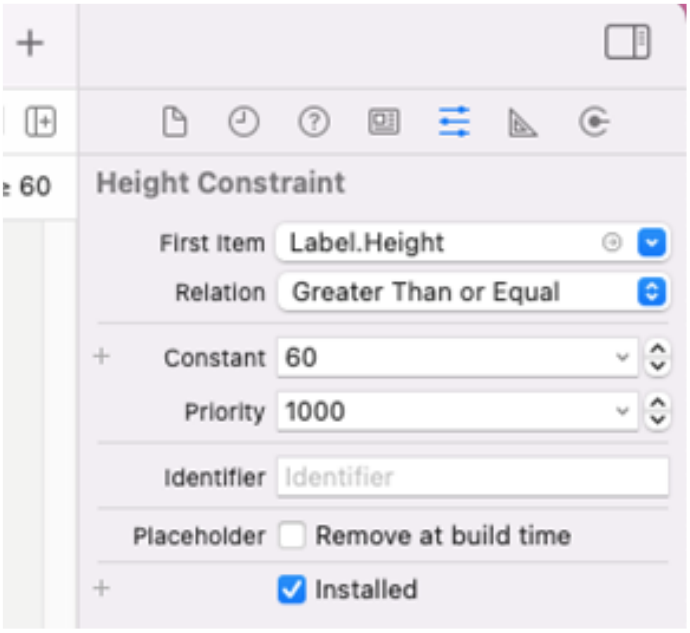
- Select the label → Size Inspector (ruler icon, open right side panel):
- Find the Height constraint → change Relation from = to ≥ and keep 60.

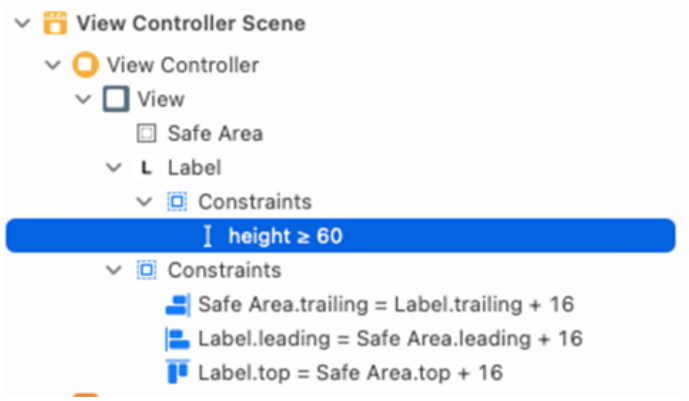


Inspect your view hierarchy

Attributes Inspector for the label:

- Text Alignment = Right
- Font ≈ 34 pt
- Lines = 1





Add a keypad container (UIView) at the bottom

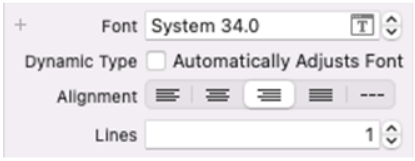
Object Library → drag a UIView below the label (this will hold the keys).

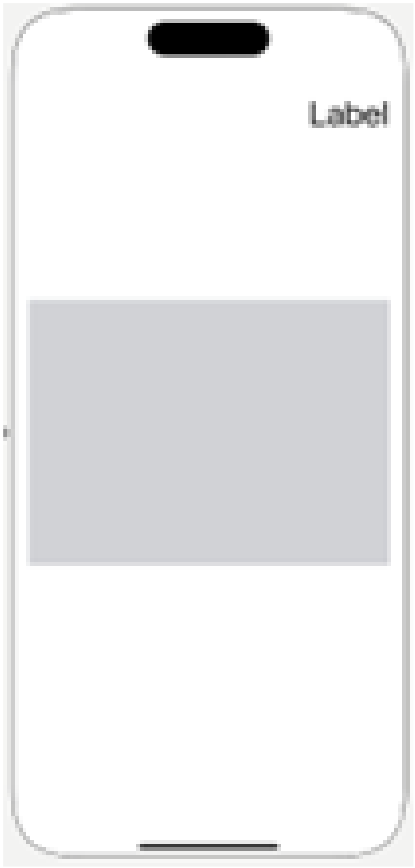
Pin constraints to the Safe Area:

- Leading = 16, Trailing = 16, Bottom = 16
- Click Add 3 Constraints.

With the container selected → Attributes Inspector:

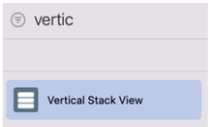
- Set Background to systemGray6 so the keypad area is visible.
- You can move the keypad up and down on the screen





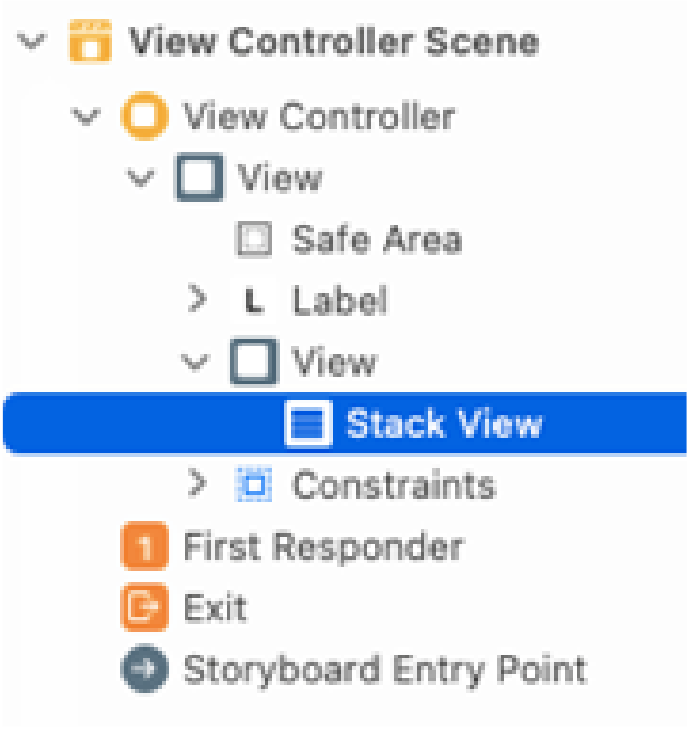
Add a vertical stack view

- Object Library → drag a Stack View into the container (drop it so it becomes the container's child).



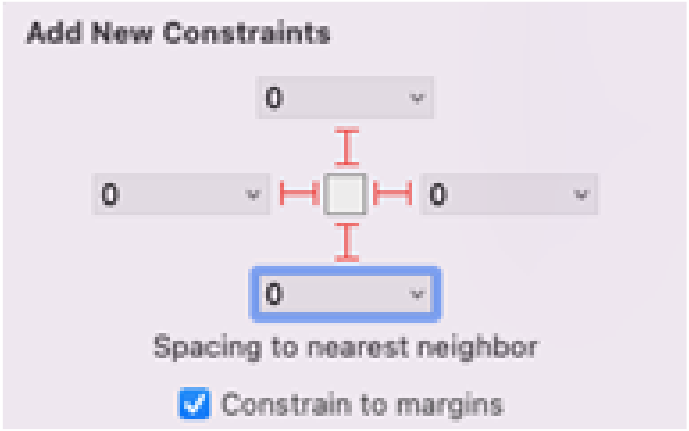
Pin the stack to the container's edges:

- Top/Leading/Trailing/Bottom = 0, then Add 4 Constraints.



With the stack selected → Attributes Inspector:

- Axis = Vertical
- Distribution = Fill Equally
- Alignment = Fill
- Spacing = 8

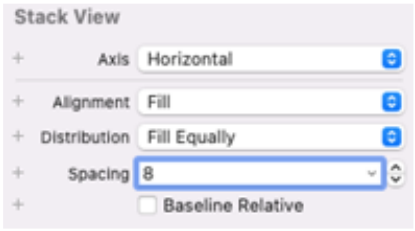


Add four horizontal stack views (rows)

- Object Library → drag a Stack View into the vertical stack.

With that row stack selected → Attributes Inspector:

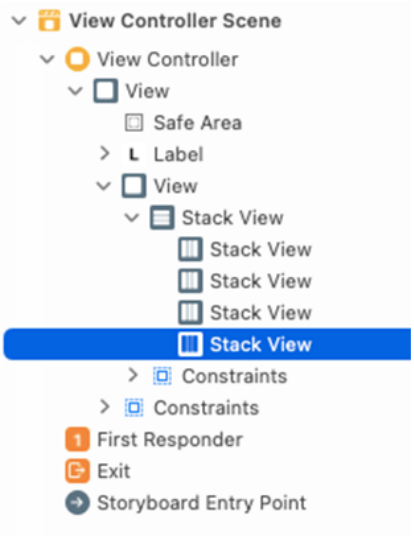
- Axis = Horizontal
- Distribution = Fill Equally
- Alignment = Fill
- Spacing = 8



Duplicate until you have 4 rows:

- EITHER drag three more horizontal stacks and set the same properties,
- OR select the first row in the Document Outline, ⌘C / ⌘V three times.

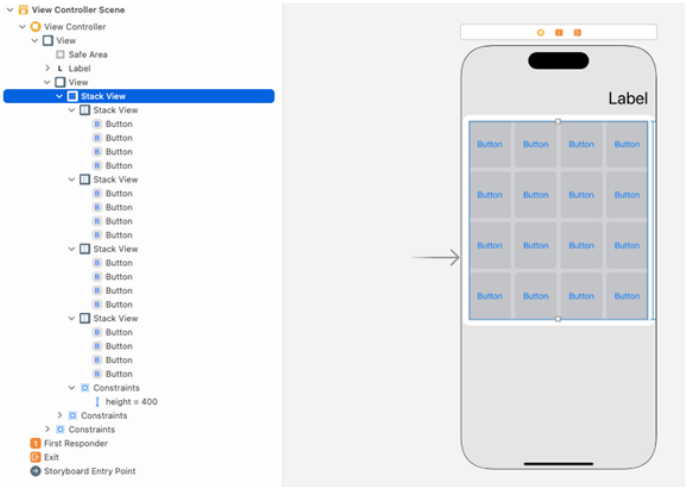
You should now have: Vertical Stack (inside container) → 4 Horizontal Stacks.



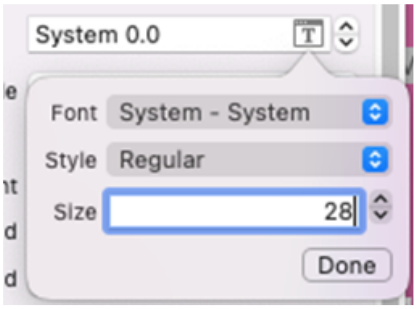
Add the 16 buttons (left → right)

For each row stack, drag Button (System or Gray) from the Object Library into that row (4 buttons per row) in the Document Outline. Set the Title of each:

Use command C and command V to copy them across one row



- Row 1: 7 8 9 ÷
- Row 2: 4 5 6 ×
- Row 3: 1 2 3 −
- Row 4: C 0 = +

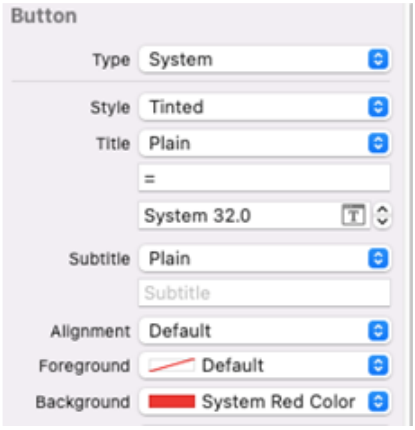


The buttons looked short so I adjusted the parent stack view height constraint to 400 to get the size above. Click each button and rename it based on the layout provided above.

Quick styling

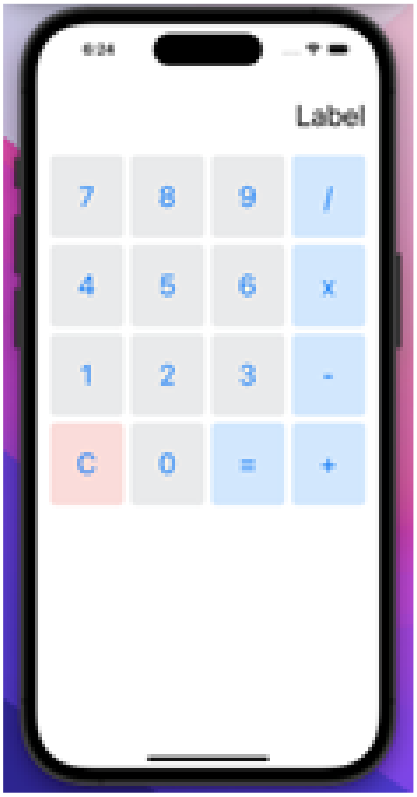
Select all digit buttons on each row and modify them:

- Set Font to System Regular and Font Size to 32

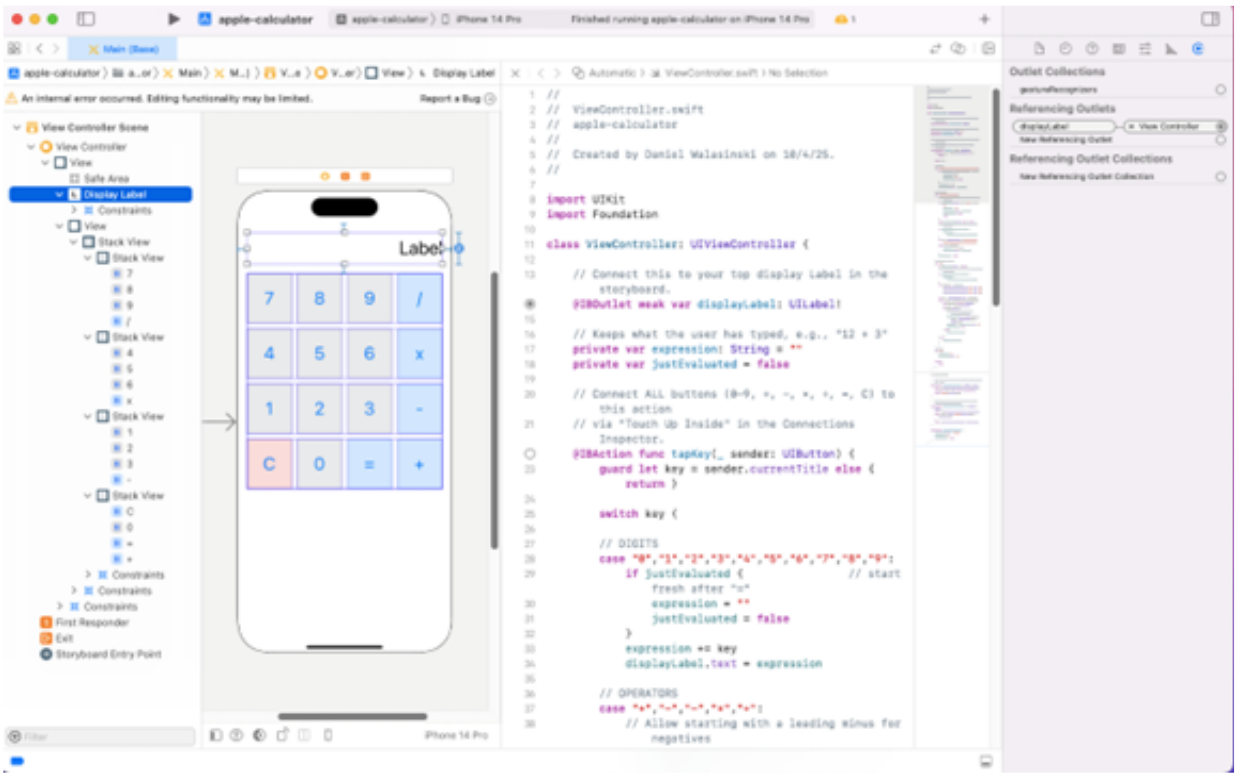


Select operators (+ − × ÷ =) → Style = Filled, Tint = System Blue.

Select C (clear) → Style = Filled, Tint = System Red.

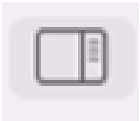


Inspect your overall design and begin making changes as needed to get your intended look and styling. The first color fix I did was go to the view that holds the buttons and changed the color to system background.



Run and view on phone simulator

- Pick a simulator from the top menu bar (e.g., iPhone 14) → Run.
- View the calculator app on your own phone to see what other adjustments may seem suitable for your design.

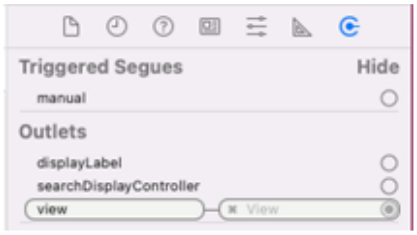


Connect UI to Code

Open the Assistant Editor, click the code icon on the top right and open Assistant. - This lets you see both the storyboard and code side by side.

I then setup my screen like this

- Control-drag from your Label to the code → make an @IBOutlet called displayLabel.



Connect the Outlet (displayLabel)

- Open Main.storyboard
- Open the right panel

Open the Connections Inspector

Under Outlets, find displayLabel — there's an empty circle next to it.

Drag from that empty circle onto your display Label (either on the canvas or in the Outline).

Release when the label highlights. You should now see displayLabel filled and showing a reference to your label.

Connect it to here

Connect each button to the code

- Under Sent Events, find Touch Up Inside.
- Drag the little circle next to Touch Up Inside onto the View Controller and release.
- Choose tapKey: from the popup.

Click through all the buttons to check if they are all connected.

Add the Swift Code

Remove the old code in the file and replace it with code for your calculator. Use the snippet below or generate code. Ensure the agent knows which version of Xcode you are running as well as the hierarchy of components you used.

If you are using an AI agent, use this prompt or ensure that your prompt includes some key information so that the AI agent knows the design of your app so that it can all be programmed, this prompt was generated by reverse engineering the code that I developed together with the agent as we completed the project in smaller bits which we may call context building.

Prompt to generate the code:

```
You are generating iOS UIKit (Storyboard) code for Xcode 14.2 (iOS 16+). The
project already has a storyboard scene set as the Initial View Controller with
this UI:One top UILabel (the “display”).A bottom keypad made with stack views
containing 16 UIButtons with titles:Row1: 7 8 9 ÷Row2: 4 5 6 ×Row3: 1 2 3 −Row4:
C 0 = +Some classes might use plain x or * instead of ×, and / instead of ÷.Your
task: output a single Swift file named ViewController.swift that implements
a basic four-function calculator with the following requirements. Output code
only (no commentary).Environment & classUIKit + Foundation only. No third-party
code.File name: ViewController.swiftimport UIKit, import FoundationDefine
class ViewController: UIViewControllerProvide one IBOutlet:@IBOutlet weak var
displayLabel: UILabel!Provide one IBAction to handle all buttons:@IBAction
func tapKey(_ sender: UIButton)Students will connect the outlet to the label,
and connect Touch Up Inside from ALL buttons to tapKey(_:).BehaviorOn launch
(viewDidLoad), set the display to "0".Maintain an internal expression: String
and a justEvaluated: Bool.Digits (0-9): append to expression; if justEvaluated
== true, start a new expression before appending.Operators: accept +, -, −, ×,
÷, x, X, *, /.Allow starting with a leading minus (- or −) to enter negative
numbers.If the last non-space character is already an operator, replace it
(prevent double operators).Display normalization: when the user taps:show ×
for any of x, X, *show ÷ for /otherwise show the tapped operatorExpression
should use spaces around operators for readability, e.g. "12 × 3".Equals
(=):If the expression ends with an operator, trim the trailing operator
before evaluation.Build a sanitized string for evaluation:×/x/X → *÷// →
```

/- → -Evaluate using NSExpression(format:).If result is finite:Show integers without .0; otherwise show the double as string.Set expression to the result so the user can chain operations (e.g., 2 + 3 = → 5, then + 4 = → 9).Set justEvaluated = true.If not finite or evaluation fails: show "Error" and clear state.Clear (C): reset state; set display to "0".Debug prints (must log to Xcode console)Add prints so beginners can confirm wiring:In viewDidLoad():print(" [App] Startup — ViewController.viewDidLoad")In viewDidAppear(_:): print a line confirming visibility (window/scene optional).At the very top of tapKey(_:): print the tapped key robustly, supporting modern UIButton configurations:let key = sender.currentTitle ?? sender.titleLabel?.text ?? sender.configuration?.title ?? "?"print(" [CalculatorVC] Tapped:", key)After equals with success: print(" [CalculatorVC] = \(text)")On clear: print(" [CalculatorVC] Cleared.")On evaluation error: print(" [CalculatorVC] Error evaluating: \"\(expression)\"")HelpersImplement these helpers (or equivalent) and use them:endsWithOperator(_:) → returns true if the trimmed expression ends with any of +-*/xX×÷dropTrailingOperator(_:) → removes one trailing operator and trims whitespace.Code style/constraintsNo SwiftUI.No additional view code (students already built the storyboard).No external dependencies.Keep everything inside ViewController.Make sure the file compiles in Xcode 14.2 with iOS 16 simulator.DeliverableOutput only the full contents of ViewController.swift meeting all requirements above. Do not include explanations, comments outside the code block, or extra files.

Or use the complete code below:

```
////  ViewController.swift//  apple-calculator////  Created by Daniel Walasinski on 10/4/25.//import UIKitimport Foundationclass ViewController: UIViewController {    // Connect this to your display Label in Interface Builder.    @IBOutlet weak var displayLabel: UILabel!    // Calculator state    private var expression: String = ""    private var justEvaluated = false    // MARK: - Lifecycle    override func viewDidLoad() {        super.viewDidLoad()        displayLabel.text = "0"        print(" [App] Startup — ViewController.viewDidLoad")    }    override func viewDidAppear(_ animated: Bool) {        super.viewDidAppear(animated)        let windowPtr = view.window.map { Unmanaged.passUnretained($0).toOpaque() }        let scenePtr  = view.window?.windowScene.map { Unmanaged.passUnretained($0).toOpaque() }       let state     = view.window?.windowScene?.activationState print(" [CalculatorVC] Appeared — window=\(String(describing: windowPtr)), scene=\(String(describing: scenePtr)), state=\(String(describing: state))")    }    // MARK: - Actions    // Connect ALL buttons (0-9, +, -, ×, ÷, x, X, *, /, =, C) to this action via "Touch Up Inside".    @IBAction func tapKey(_ sender: UIButton) {       let key = sender.currentTitle          ?? sender.titleLabel?.text          ?? sender.configuration?.title           ?? "?"        print(" [CalculatorVC] Tapped:", key)        switch key {        // DIGITS        case "0","1","2","3","4","5","6","7","8","9":            if justEvaluated {                expression = ""                justEvaluated = false            }            expression += key            displayLabel.text = expression        // OPERATORS: accept +, -, –, ×, ÷, x, X, *, /        case "+","-","–","×","÷","x","X","*","/":            // allow starting with a leading minus for negatives            if expression.isEmpty, key == "-" || key == "-" {                expression = "-"                displayLabel.text = expression                justEvaluated = false                return            }            guard !expression.isEmpty else { return }            // If the last char is already an operator, replace it            if endsWithOperator(expression) {                expression = dropTrailingOperator(expression)            }            // Normalize how we SHOW the operator:            //  x, X, *  → "×"            //  /        → "÷"            //  others   → as-is            let displayOp: String            switch key {            case "x","X","*": displayOp = "×"            case "/":         displayOp = "÷"            default:          displayOp = key            }            expression += " \(displayOp) "            displayLabel.text = expression
```

```
justEvaluated = false        // EQUALS — evaluate expression        case "=":            guard
!expression.isEmpty else { return }           if endsWithOperator(expression) {               expression
= dropTrailingOperator(expression)            }            // Map all visuals/variants to computer math
for NSExpression           let sanitized = expression               .replacingOccurrences(of: "×", with:
"*")              .replacingOccurrences(of: "x", with: "*")              .replacingOccurrences(of: "X",
with: "*")              .replacingOccurrences(of: "÷", with: "/")              .replacingOccurrences(of:
"-", with: "-")            let expr = NSExpression(format: sanitized)            if let number =
expr.expressionValue(with: nil, context: nil) as? NSNumber {               let value = number.doubleValue
if value.isFinite {                    let text = (floor(value) == value) ? String(Int(value)) :
String(value)                   displayLabel.text = text                   expression = text          //
allow chaining (result then operator)                   justEvaluated = true                   print("
[CalculatorVC] = \(text)")               } else {                   showErrorAndReset()               }
} else {              showErrorAndReset()          }       // CLEAR       case "C":           expression
= ""           displayLabel.text = "0"           justEvaluated = false           print(" [CalculatorVC]
Cleared.")       default:           print(" [CalculatorVC] Unknown key title: \"\(key)\"")       }    }
// MARK: - Helpers    /// True if the trimmed expression ends with any operator variant.    private func
endsWithOperator(_ s: String) -> Bool {       let t = s.trimmingCharacters(in: .whitespaces)       guard
let last = t.last else { return false }        return "+-*/xX×÷".contains(last)    }    /// Drops one
trailing operator (and extra space), returns cleaned string.    private func dropTrailingOperator(_ s:
String) -> String {        var t = s.trimmingCharacters(in: .whitespaces)        if let last = t.last,
"+-*/xX×÷".contains(last) {            t.removeLast()        }        return t.trimmingCharacters(in:
.whitespaces)    }    private func showErrorAndReset() {        print(" [CalculatorVC] Error evaluating:
\"\(expression)\"")       displayLabel.text = "Error"       expression = ""       justEvaluated = false
}}
```

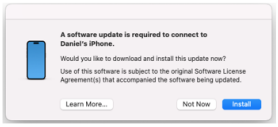
Run the App

At the top left of Xcode, click the Run ▶ button.2. Your calculator opens in a new window — test out all the functions. If you press buttons and nothing is displayed on the screen, then you should go back to step 4 and check that all your buttons are connected. If they are connected check the debug window by opening it at the bottom. Run the app and press the buttons. The code above will post messages in the debug window so that I can confirm they are working. If they are working then the problem is with the display or calculator code. When you make an app ensure you add debugging information to your buttons when needed or if you're stuck.

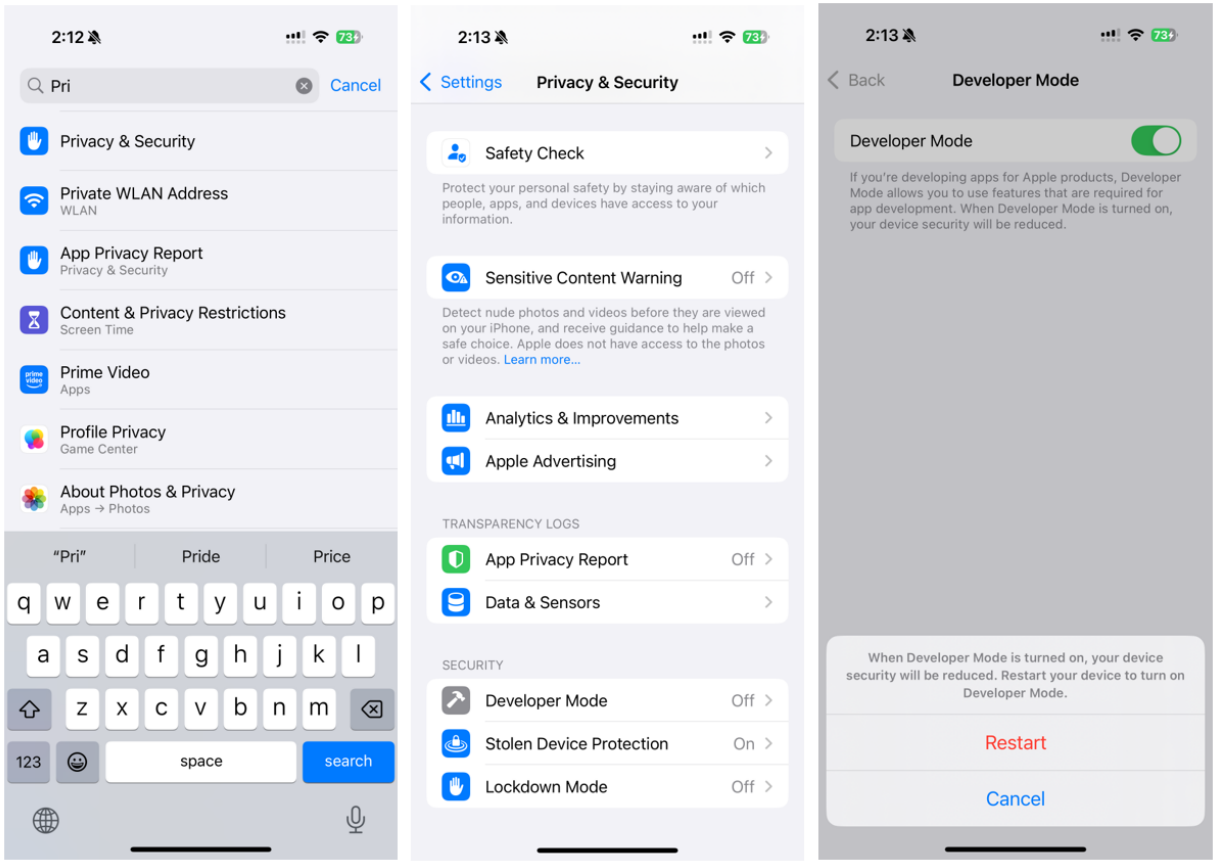
Run on your iPhone

Plug in your iPhone and tap Trust This Computer. (I used an older non-iPhone USB-C to USB cable). My phone showed a message: Trust this computer? Select yes.

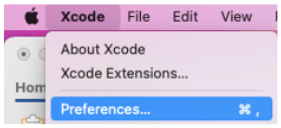
- Then my computer installed some updates.

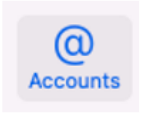

A software update is required to connect to Daniel's iPhone.
Would you like to download and install this update now?
Use of this software is subject to the original Software License Agreement(s) that accompanied the software being updated.
Learn More...
Not Now
Install

On iPhone (iOS 16+): Settings ▸ Privacy & Security ▸ Developer Mode → On (phone restarts).

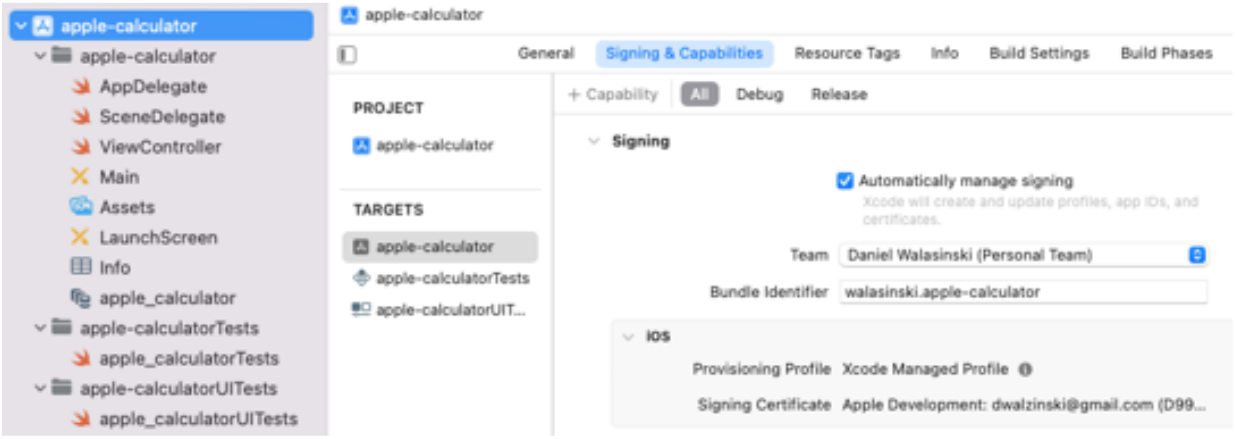


Check if your logged into the same Apple ID inside of Xcode. In Xcode find the top left menu button: Xcode ▸ Preferences ▸ Accounts ▸ → sign in with your Apple ID.

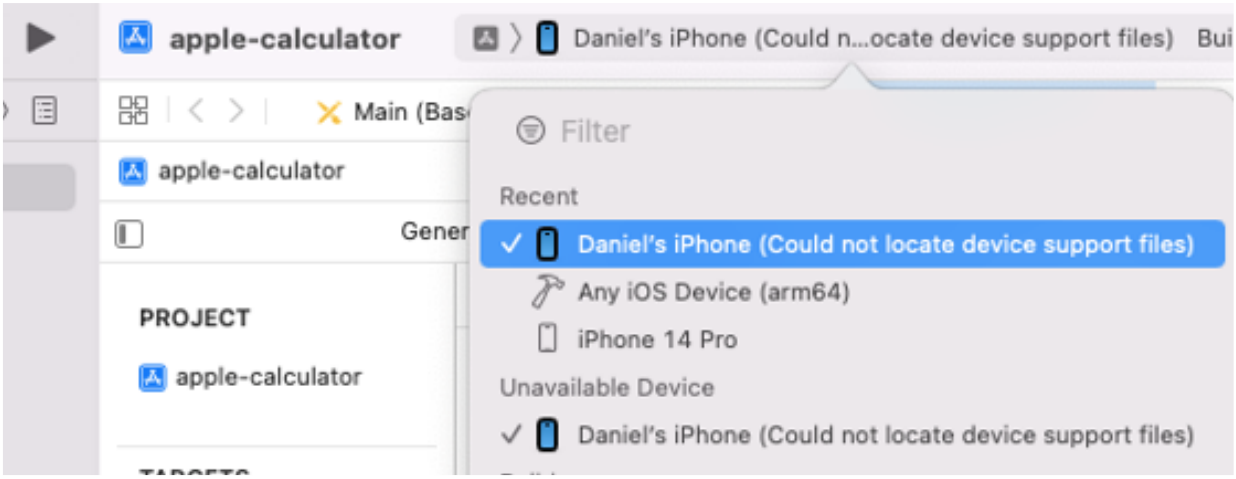




Inside Xcode click you main Application file, on the right side menu choose signing & capabilities. Enable Autoamtically manage signing, choose your personal team or create one, make sure you have a bundle identifier.



Toolbar device = Your iPhone → press Run .



Older versions of Xcode won't be able to run apps on phones, therefore if you are using an older Mac computer or operating system, you'll need to upgrade that to share, publish and test apps further.

The app installs and launches on your phone. (Next time you can enable Connect via network to run over Wi-Fi.)

Now you can test your app and show others your app on your phone. For next steps you will need to sign up for an Apple Developer account for $100 per year.

Why some developers choose Storyboard instead of SwiftUI

Storyboard is a visual interface design system inside Xcode that represents screens and UI components as objects on a canvas, with connections to Swift code through outlets and actions. Developers may choose Storyboard when they want direct control over UIKit components, need to work on an existing codebase that already uses Storyboards, or prefer building complex layouts with Interface Builder and Auto Layout without writing the initial layout code by hand. Storyboards also make it easy to inspect a view hierarchy, adjust constraints visually, and verify that elements are connected correctly, which can be helpful for beginners and for teams that use a designer-developer workflow within Xcode.

Chapter 11 Make an App for iPhone on Mac Using Storyboard Rubric

This chapter focused on creating an iPhone app in Xcode using Storyboard, connecting interface elements to Swift code, testing the app in the simulator or on a device, revising problems, and documenting the development process.

Category	Points	Evidence of Completion
Project Setup	5	An Xcode iPhone app project is created, clearly named, saved, and organized with the required Storyboard and Swift files.
Storyboard Interface	5	The app interface is created in Storyboard with a clear layout, display area, buttons, labels, stack views, or other required screen elements.
Swift Code and Connections	5	Interface elements are connected to Swift code using outlets, actions, or other required connections. The app includes working logic for the required features.
Testing and Revision	5	The app is tested in the iPhone simulator or on a device. Layout issues, broken connections, button errors, or functionality problems are identified and revised.
Evidence of Work	5	Screenshots, Storyboard evidence, Swift code evidence, simulator or device testing proof, or other project evidence are submitted.
Reflection	5	A short explanation describes the iPhone app, how Storyboard and Swift worked together, what was tested or revised, and relevant computer science vocabulary such as Storyboard, Swift, outlet, action, interface, button, function, simulator, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
Xcode iPhone app project created	☐
Project clearly named and saved	☐
Storyboard interface completed	☐
Display area, buttons, or required screen elements included	☐
Interface elements connected to Swift code	☐
Required app functionality completed	☐
App tested in simulator or on a device	☐
At least one layout issue, connection issue, or functionality problem revised	☐
Screenshots or project proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the iPhone app created in Xcode. Explain how Storyboard and Swift were used together to build the interface and functionality. Identify one layout issue, connection issue, or functionality problem that was tested and improved before final submission.*

12
Build a Mobile App with VS Code + Expo

This lesson explains how to build and test a cross platform mobile app using Visual Studio Code and the Expo toolchain. You will install the required tools, create a new Expo project from the terminal, run the development server, and open the app on an iPhone or Android device using Expo Go. The lesson also introduces the core workflow used in modern app development: installing dependencies, editing code files, restarting the development server when needed, and using error messages to guide troubleshooting.

Install Node.js – JavaScript runtime

- Go to: https://nodejs.org and install it.
- Verify: open a terminal
- Windows: Press Win key → type cmd → Enter
- Mac: Press Command ⌘ + Space → type Terminal → Enter

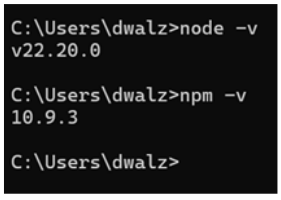
Run JavaScript Everywhere
Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.
Get Node.js®

Run

Command:

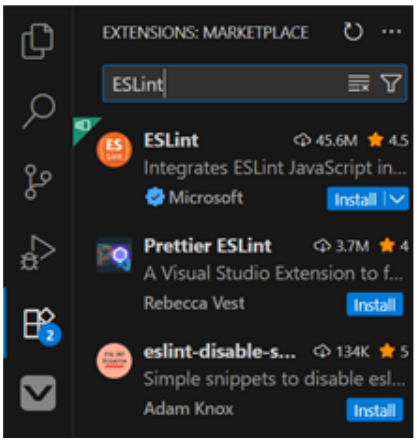
```
node -vnpm -v
```

You should see version numbers.

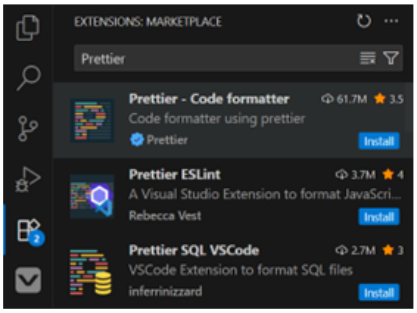


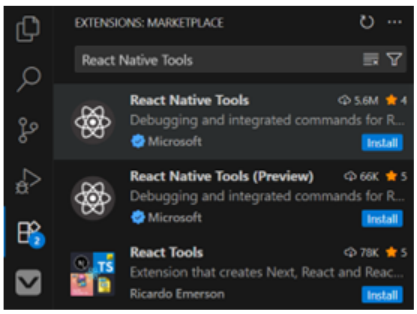
Download or Open Visual Studio Code

- Go to https://code.visualstudio.com/
- Download and install.
- Open VS Code → click Extensions (square icon on the left). Install:
- ESLint by Microsoft
- Prettier by Prettier



React Native Tools by Microsoft





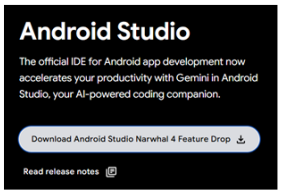
Download or open Expo Go on your phone

- iPhone: App Store → search Expo Go
- Android: Google Play → search Expo Go

Emulators/Simulators

You may also use your physical iOS or Android devices.

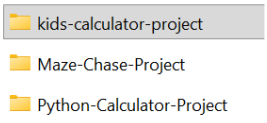
- Install Android Studio https://developer.android.com/studio. Install Android Virtual Device.
- Install Xcode from the App Store on MacBook for iOS Simulator.


Android Studio
The official IDE for Android app development now accelerates your productivity with Gemini in Android Studio, your AI-powered coding companion.
Download Android Studio Narwhal 4 Feature Drop
Read release notes

Create your first Expo app

Open VS Code.

Go to File, click open folder, make a project folder and name it your app name plus project (no spaces use caps or dashes), press select folder. If Augment tries to index your codebase decline it, open Extensions and uninstall it. You should have your App folder ready and the Terminal open.



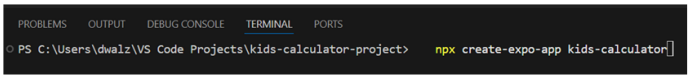
In the Terminal run this commands, change (name-your-app) to your app name no spaces.

Command:

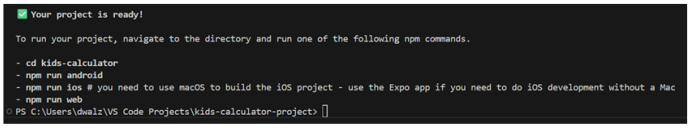
```
npx create-expo-app (name-your-app)
```

Press enter

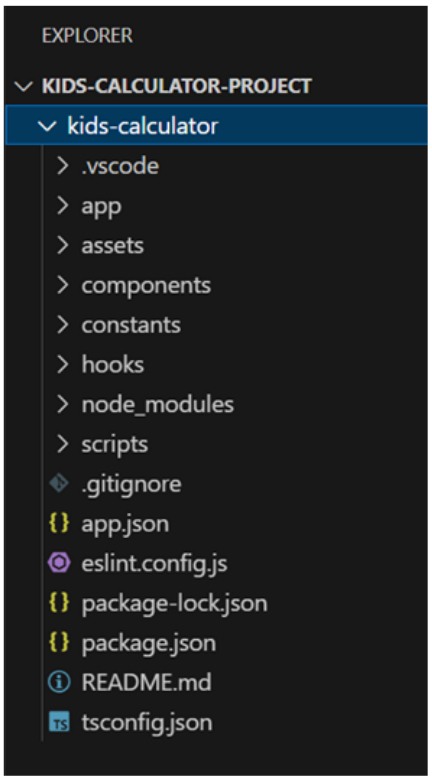
Wait a couple minutes for it to finish



When finished you'll get this message and your project files and folders will be prepared.


Your project is ready!
To run your project, navigate to the directory and run one of the following npm commands.
- cd kids-calculator
- npm run android
- npm run ios # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development without a Mac
- npm run web
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project>

Now if you open your app folder you'll find new folders and files you installed.



Next type this into the terminal, replace (your-app-name) with the name of your app or the app folder you just created.

Command:

```
cd (your-app-name)
```

Then type:

Command:

```
npm start
```

or

Command:

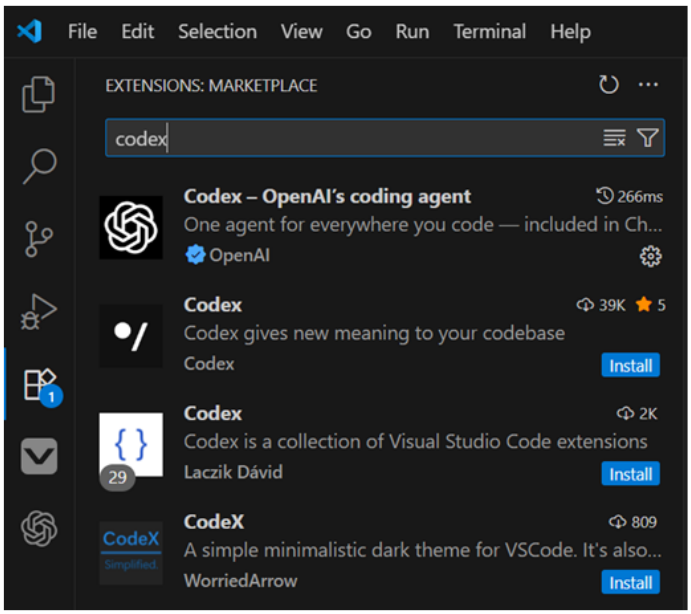
```
npx expo start
```

This starts the Expo development server and shows a QR code.

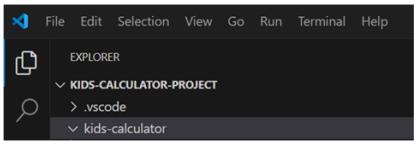
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator> npm start
Starting project at C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator
React Compiler enabled
Starting Metro Bundler
› Metro waiting on exp://192.168.0.103:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
› Web is waiting on http://localhost:8081
› Using Expo Go
› Press s | switch to development build
› Press a | open Android
› Press w | open web
› Press j | open debugger
› Press r | reload app
› Press m | toggle menu
› shift+m | more tools
› Press o | open project code in your editor
› Press ? | show all commands
Logs for your project will appear below. Press Ctrl+C to exit.

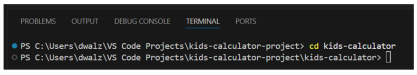
Code your App

I ended up importing my files from the calculator app I build earlier and then used Open AI's Code-ex to fix, add, or adjust files as needed in order to get it working. Additionally you can use Codex inside of VScode to build your entire project similar to how you would in Cursor. Because this agent will be running code in your terminal, it is sometimes recommended to create a separate Windows user account while using AI to code locally on your computer so that if anything were to get broken or corrupt, it wouldn't affect your main user account.



In the terminal target the sub folder that holds the app with cd (app folder)




PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project> cd kids-calculator
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>

Because I imported files and relaced some I need to run this again.

Command:

```
npm install
```

and

Command:

```
npx expo start
```

If you are already logged into your expo App on your phone the project will appear there. If not scan the QR code to try. If it doesn't work cycle the apps and the above npx expo start in a new terminal.

I tested my app using the Expo go app on my iPhone and Google Pixel. You could also test it out on a tablet too. If you don't have an android phone you can press a to open an emulator, follow the error message https://docs.expo.dev/workflow/android-studio-emulator to learn how to set up a virtual environment for it.

- Test the functionality and make any final edits.

Final configuration

In VS Code, open the file app.json (or app.config.ts if that is what the project uses).

Find and set these fields. Edit them if needed. Some actions below may be completed by your AI agent but you should check manually as well and prepare an icon image too.

- **App name**

This is what shows under the icon on the phone. Keep it short. Search for apps on the app store and try to make a unique name for yours.

Code:

```
"name": "Kids Calculator"
```

- **Slug**

This is the internal project name for Expo. No spaces.

Code:

```
"slug": "kids-calculator"
```

- **Icon**

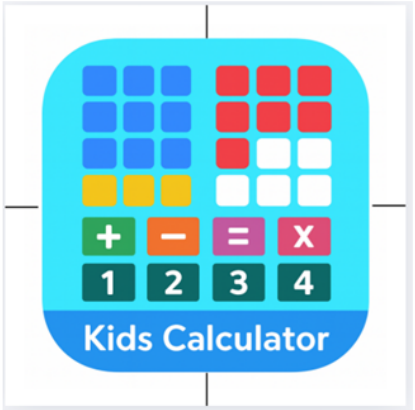
This points to the app icon file in your project. If you didn't create an icon make a 1024 × 1024 pixel square PNG file using Figma or Canva etc and manually drop your file in that location.

Code:

```
"icon": "./assets/icon.png"
```

I made mine in Canva and you'll want to distance the sides 80 to 100 px away from the edge. I measured this by adding 85 px lines and centering the icon, making sure to preserve the same px for width and height of the icon. Save as a PNG, a transparent background will not be needed when for the mobile app, just remove your measurement lines.

Update: After publishing the app I noticed there is white space around the outside of the app icon, therefore you may want to reduce the white space around the sides to 0 px.



Action: Make sure that file actually exists in assets/. Again you'll want to research splash screens and make one appropriate for you.

Splash screen

This is the first screen users see when the app loads. Use a canvas this size: 1242×2436 px, you can put your icon in the center and change the rest of the background to your brand's color instead of white. (in google search splash screen examples).

Code:

```
"splash": {
```

Code:

```
"image": "./assets/splash.png",
```

Code:

```
"resizeMode": "contain",
```

Code:

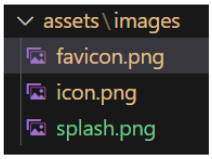
```
"backgroundColor": "#ffffff"
```

Code:

```
}
```

Action: Make sure splash.png exists.

I removed the other placeholder files and just have these three asset images, I added a favicon by resizing the icon in Canva to 512 by 512 but will only be visible if this app is also deployed as a web pp.



Inside app.json need to update version numbers and add bundle identifiers required for publishing.

Code:

```
"version": "1.0.0",
```

Code:

```
"ios": {
```

Code:

```
"bundleIdentifier": "(firstnamelastname).kidscalculator",
```

Code:

```
"buildNumber": "1"
```

Code:

```
},
```

Code:

```
"android": {
```

Code:

```
"package": ""(firstnamelastname).kidscalculator",
```

Code:

```
"versionCode": 1
```

Code:

```
}
```

Rules:

bundleIdentifier and package must stay the same forever for this app.

Only change the numbers when you upload a new update later.

Orientation

Lock the screen if your app should only be portrait.

Code:

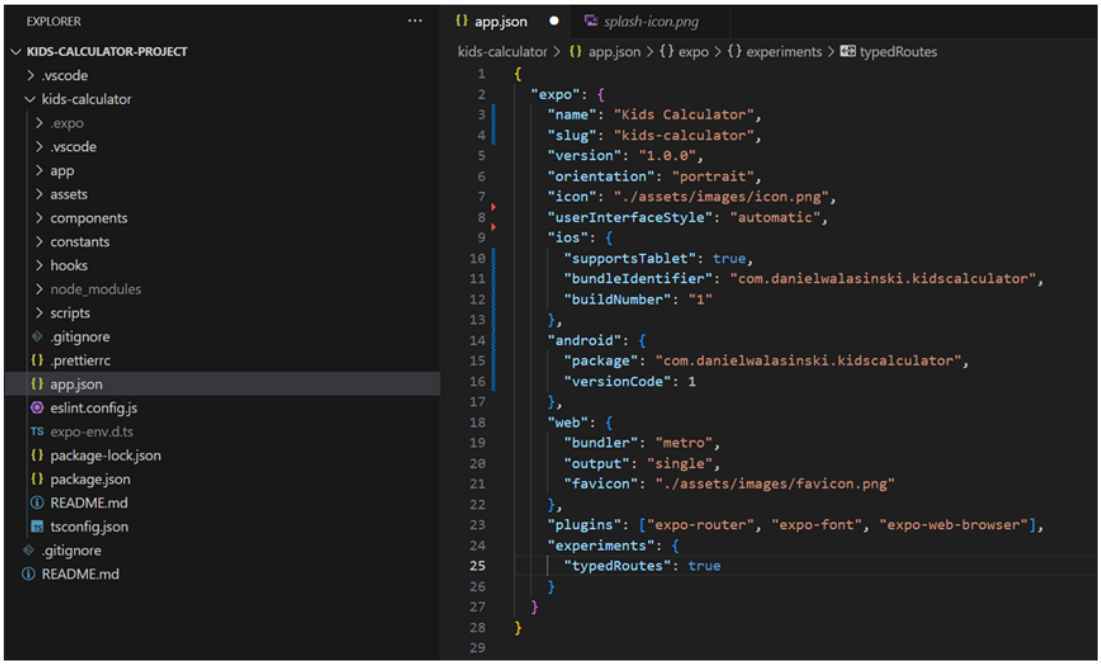
```
"orientation": "portrait"
```

Permissions

If your app does not need camera, mic, location, etc, then do not ask for them and it doesn't need to be written in your code, otherwise it will be written inside of the app.json file.

In the android section you can keep "permissions": [] if you do not need anything special.

Save the file.

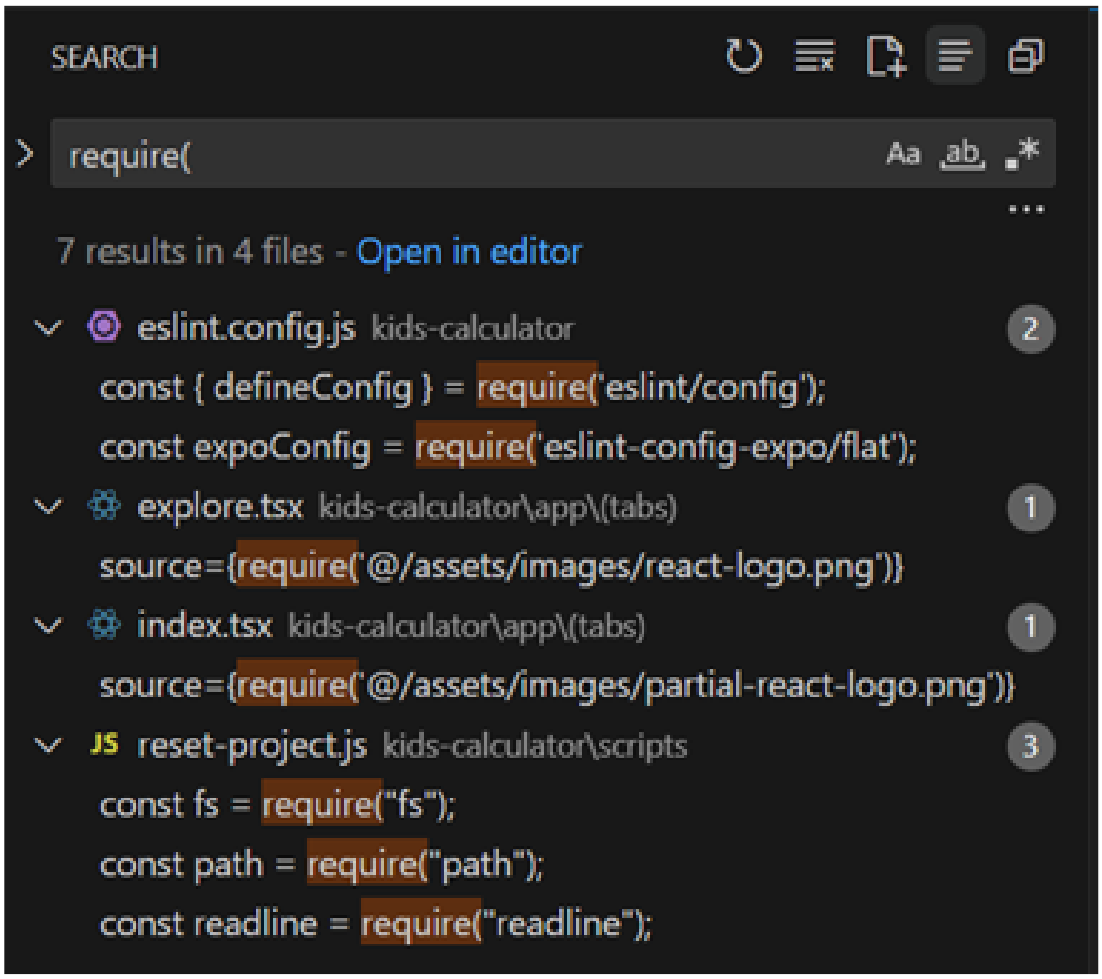


Check your assets

Make sure images, logos, fonts, and sounds are all real and load correctly. Missing assets can crash the app.

How to do it

In VS Code press control shift F to find and search for require(



I then uploaded this image to my AI agent to have a look. Click each result from your search. Then highlight the explore.tsx and index.tsx and delete those files by clicking the x.

Delete any huge images or test images that the app does not use.

Fix screens and navigation

Goal

Make sure every screen in the app can load. Make sure tapping around the app does not break. My app only has one screen so I didn't need them.

How to do it

In VS Code open each file under app/.

Example:

app/index.tsx

app/(tabs)/explore.tsx

app/(tabs)/home.tsx

At the bottom (or near the bottom) of each file you should see something like:

Code:

```
export default function ExploreScreen() {
```

Code:

```
return (
```

Code:

```
<View>
```

Code:

```
<Text>Explore screen</Text>
```

Code:

```
</View>
```

Code:

```
);
```

Code:

```
}
```

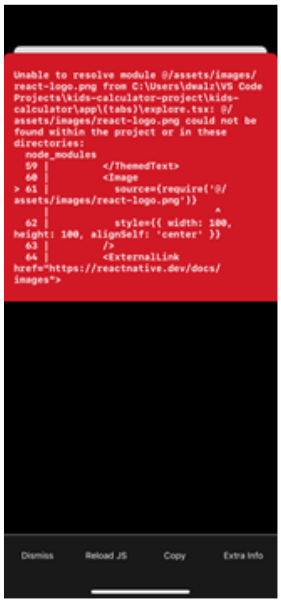
Open it on a phone using Expo Go

In the terminal type:

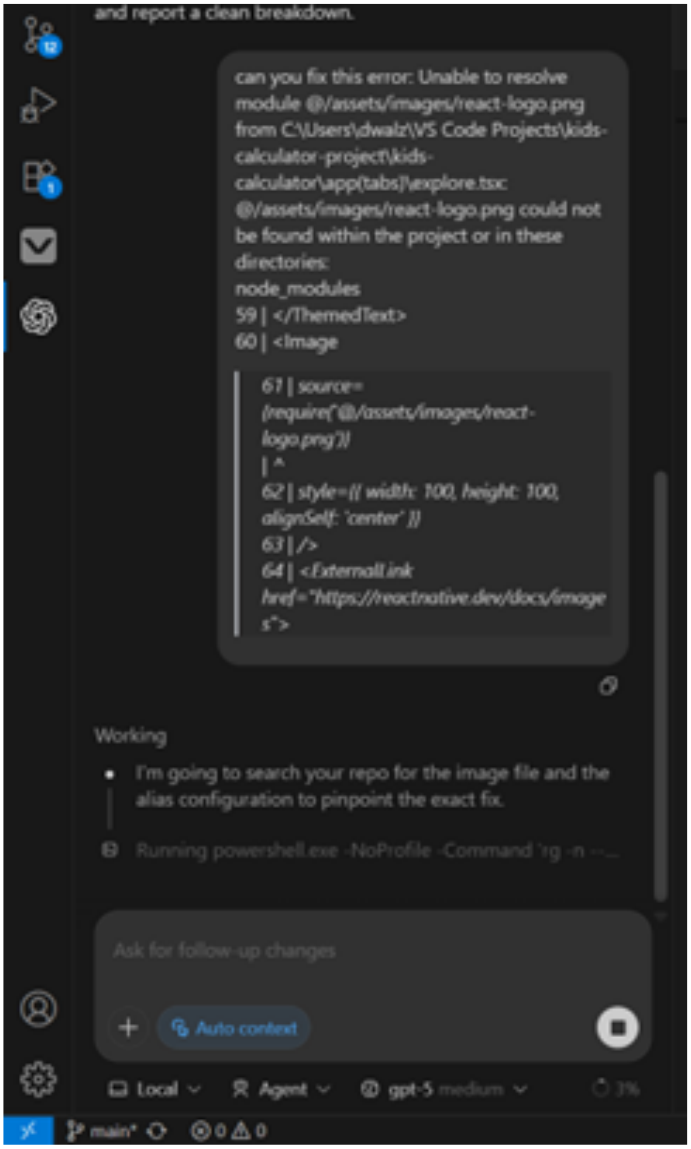
```
npx expo start
```

Then open it on a phone using Expo Go.

Tap every tab and every main button. The app should not crash or show a big red error screen. If it does, fix that screen.



If you get an error copy it and paste it into your AI agent inside VS code.



Watch what it is doing and approve all of the changes. The AI agent will clean up the code and iron out any wrinkles. If instead you didn't want to edit the code manually you can enter the commands above to edit or remove code but for some things you'll need to add them in manually.

After the changes restart the Metro bundler and test again with the QR code.

Press control c to leave the Expo process

Command:

```
npx expo start -clear
```

If you get this error and the app still loads it can be fixed by doing the following.

In your terminal:

`Control c`

Command:

```
npm install -g eas-cli
```

then:

Command:

```
eas login
```

login with your EXPO username and password inside of the terminal and say yes to all the prompts until it asks if you have an Apple Developer Account, if you haven't set it up yet press no for that one, you'll need to do that later to get your .ipa file which is the standard iOS app package.

Control c to exit

Clean up dependencies

Goal

Make sure the libraries your app uses are correct for the version of Expo you are on. This prevents build failures.

How to do it

In VS Code terminal (Terminal tab at the bottom), run:

Command:

```
npm install
```

Command:

```
npx expo-doctor
```

Watch the output. If it says there is an error with a package version, fix it.

Use:

Command:

```
npx expo install --check
```

Example: if react-native-worklets is the wrong version, install the version Expo expects:

Command:

```
npm install react-native-worklets@0.5.1
```

Run:

Command:

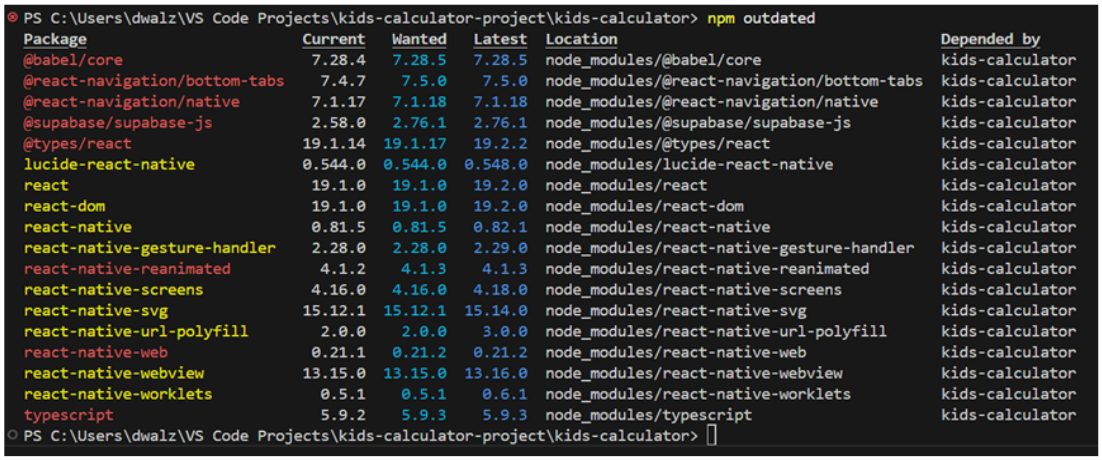
```
npx expo-doctor
```

Do you have 0 vulnerabilities? If yes then continue

Run:

Command:

```
npm outdated
```


PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator> npm outdated
Package Current Wanted Latest Location Depended by
@babel/core 7.28.4 7.28.5 7.28.5 node_modules/@babel/core kids-calculator
@react-navigation/bottom-tabs 7.4.7 7.5.0 7.5.0 node_modules/@react-navigation/bottom-tabs kids-calculator
@react-navigation/native 7.1.17 7.1.18 7.1.18 node_modules/@react-navigation/native kids-calculator
@supabase/supabase-js 2.58.0 2.76.1 2.76.1 node_modules/@supabase/supabase-js kids-calculator
@types/react 19.1.14 19.1.17 19.2.2 node_modules/@types/react kids-calculator
lucide-react-native 0.544.0 0.544.0 0.548.0 node_modules/lucide-react-native kids-calculator
react 19.1.0 19.1.0 19.2.0 node_modules/react kids-calculator
react-dom 19.1.0 19.1.0 19.2.0 node_modules/react-dom kids-calculator
react-native 0.81.5 0.81.5 0.82.1 node_modules/react-native kids-calculator
react-native-gesture-handler 2.28.0 2.28.0 2.29.0 node_modules/react-native-gesture-handler kids-calculator
react-native-reanimated 4.1.2 4.1.3 4.1.3 node_modules/react-native-reanimated kids-calculator
react-native-screens 4.16.0 4.16.0 4.18.0 node_modules/react-native-screens kids-calculator
react-native-svg 15.12.1 15.12.1 15.14.0 node_modules/react-native-svg kids-calculator
react-native-url-polyfill 2.0.0 2.0.0 3.0.0 node_modules/react-native-url-polyfill kids-calculator
react-native-web 0.21.1 0.21.2 0.21.2 node_modules/react-native-web kids-calculator
react-native-webview 13.15.0 13.15.0 13.16.0 node_modules/react-native-webview kids-calculator
react-native-worklets 0.5.1 0.5.1 0.6.1 node_modules/react-native-worklets kids-calculator
typescript 5.9.2 5.9.3 5.9.3 node_modules/typescript kids-calculator
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>

Paste it into your external AI agent to see if any of these are required to be updated, although some of mine were out of date, it wasn't recommended.

Perform a smoke test on your devices

Prove the app can open, show the first screen, and navigate without crashing on both Android and iPhone.

How to do it

In the VS Code terminal run:

```
j
```

On Android phone:

Install “Expo Go” from Play Store.

Scan the QR code.

The app should load.

On iPhone:

Install “Expo Go” from App Store.

Scan the QR code.

The app should load.

Shake your device to open the EXPO Developer Menu to inspect the app as well. Disable

Check these things:

The splash screen appears then goes away. Mine has not been displaying so I had my AI agent inside of VS code explore that issue. It says that it should work once the app is exported, so we can explore this issue later as well as other small details to improve overall user experience.

Check:

The main screen text is readable.

Buttons respond when tapped.

You can go back to the main screen easily.

No “DEV ONLY,” “TODO,” or curse words in visible text.

No red error boxes.

If anything crashes, fix that code now. If it crashes now it will crash for Apple and Google reviewers. They will reject it.

Add a simple info/support screen

Goal

Review teams want to see that users can get help and see your policy.

How to do it

Make (or edit) a screen like SettingsScreen or AboutScreen.

On that screen show at least:

- App Name

- App Version (match "version" in app.json, for example Version 1.0.0)
- Support Email (for example support@yourdomain.com)
- Privacy Policy link (a URL where your privacy policy will live)

Example simple component:

Code:

```
import { View, Text, Linking, Pressable } from "react-native";
```

Code:

```
export default function InfoScreen() {
return (
```

Code:

```
<View style={{ padding: 20 }}>
```

Code:

```
<Text>Kids Calculator</Text>
```

Code:

```
<Text>Version 1.0.0</Text>
```

Code:

```
<Text>Support: support@yourdomain.com</Text>
```

Code:

```
<Pressable onPress={() => Linking.openURL("https://yourdomain.com/privacy")}>
```

Code:

```
<Text style={{ color: "blue" }}>Privacy Policy</Text>
```

Code:

```
</Pressable>
```

Code:

```
</View>
);
```

Code:

```
}
```

Make sure you can reach this screen from inside the app. Example: add it as a tab or add a button “Info / Support” on the home screen.

Reason: Apple and Google both ask for contact and privacy info. If it is inside the app, review goes smoother.

Match screenshots to the real app

Goal

The screenshots you upload to the stores must look like the app people will actually see when they open it.

How to do it

Decide what the “main screen” of the app should look like. Example: Your calculator screen.

Make sure the text, colors, button names on that screen are final. No mock data.

Run the app on a phone with npx expo start.

Take real screenshots on the phone of:

Home screen

Main feature screen

Settings / Info screen

Save these screenshots in a folder in your project, for example:

branding/screenshots/android/ and branding/screenshots/ios/.

Build the Android file (AAB)

Needed for publishing on Google Play.

Open VS Code terminal.

Make sure you’re in your app folder.

Run:

Command:

```
npm install -g eas-cli
```

Command:

```
eas login (enter your username and password for Expo go)
```

```
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>  eas login
You are already logged in as walamandan.
✓ Do you want to continue? ... yes
Log in to EAS with email or username (exit and run eas login --help to see other login options)
✓ Email or username ... dwalzinski@gmail.com
✓ Password ... *********
Logged in
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>  eas build -p android
Resolved "production" environment for the build. Learn more: https://docs.expo.dev/eas/environment-variables/#setting-the-environment-for-your-builds
No environment variables with visibility "Plain text" and "Sensitive" found for the "production" environment on EAS.

No remote versions are configured for this project, versionCode will be initialized based on the value from the local project.
✓ Incremented versionCode from 1 to 2.
android.versionCode field in app config is ignored when version source is set to remote, but this value will still be in the manifest available via expo-
's recommended to remove this value from app config.
✓ Using remote Android credentials (Expo server)
✓ Generate a new Android Keystore? ... yes
✓ Created keystore
```

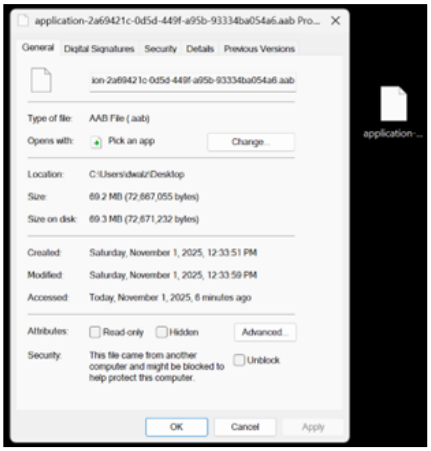
Command:

```
eas build -p android --profile production
```

Wait for Expo to finish the cloud build. (It took mine about 15 minutes)

```
Waiting for build to complete. You can press Ctrl+C to exit.
✓ Build finished

🤖 Android app:
https://expo.dev/artifacts/eas/6eKDhVudSSqubDnrUTkSaL.aab
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>
```

It will give you a download link → download the .aab.

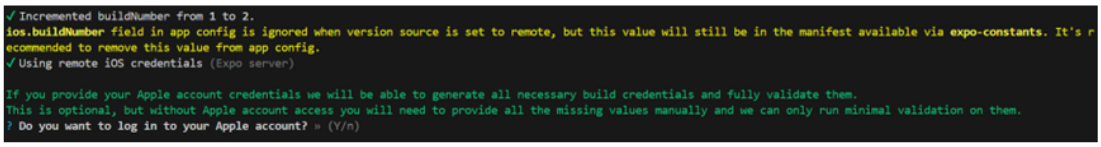


Save it in a folder called builds/android/ so you can find it later.

(If you want to test on your phone first, also build an APK:)

Command:

```
eas build -p android --profile preview
```


✓ Incremented buildNumber from 1 to 2.
ios.buildNumber field in app config is ignored when version source is set to remote, but this value will still be in the manifest available via expo-constants. It's r
ecommended to remove this value from app config.
✓ Using remote iOS credentials (Expo server)
If you provide your Apple account credentials we will be able to generate all necessary build credentials and fully validate them.
This is optional, but without Apple account access you will need to provide all the missing values manually and we can only run minimal validation on them.
? Do you want to log in to your Apple account? » (Y/n)

Build the iOS file (IPA)

Needed for publishing on Apple.

In VS code, kill the terminal and open a new one

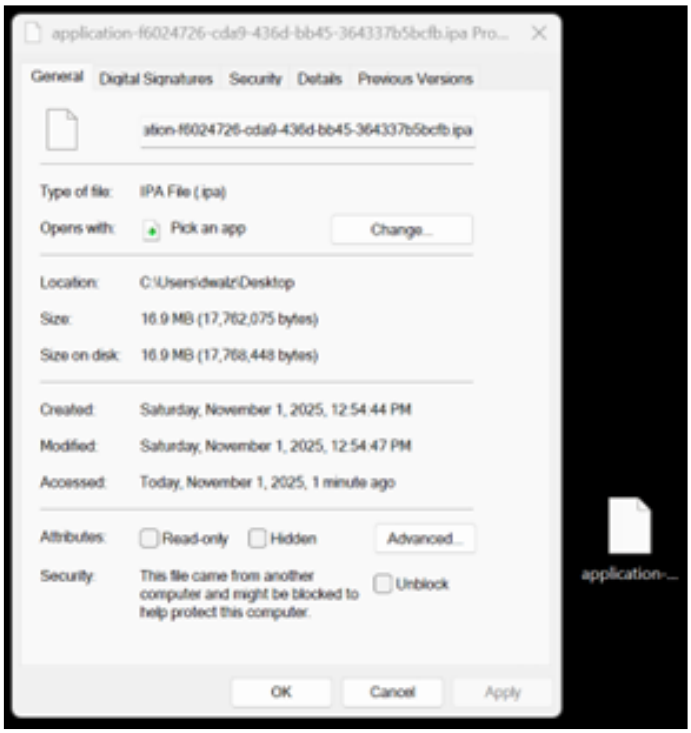
Command:

```
eas build -p ios --profile production
```

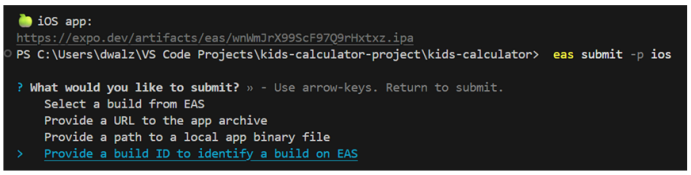
Log in to your Apple Developer account when Expo asks.

Click yes for Generate a new Apple Distribution Certificate And Generate a new Apple Provisioning Profile

Copy and save the Project Credentials Configuration to a word or text file.



When it finishes, download the .ipa.


iOS app:
https://expo.dev/artifacts/eas/wnWmJrX99ScF97Q9rHxtxz.ipa
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator> eas submit -p ios
? What would you like to submit? » - Use arrow-keys. Return to submit.
Select a build from EAS
Provide a URL to the app archive
Provide a path to a local app binary file
> Provide a build ID to identify a build on EAS

Then send it to Apple

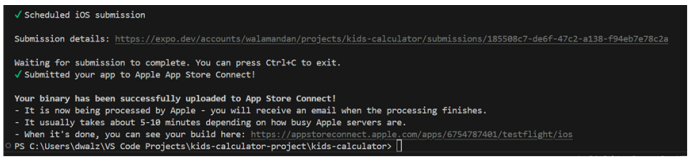
Command:

```
eas submit -p ios
```

Choose: Provide a build ID to identify a build on EAS (by using arrow keys)

Enter your build ID, it can be difficult to find, look about 7 lines
up you should see a message in the terminal that says, See logs:
https://expo.dev/accounts/walamandan/projects/kids-calculator/builds/f6024726-cda9-436d-bb45-364337b5bcfb

The build ID is the number I highlighted in red, find yours.

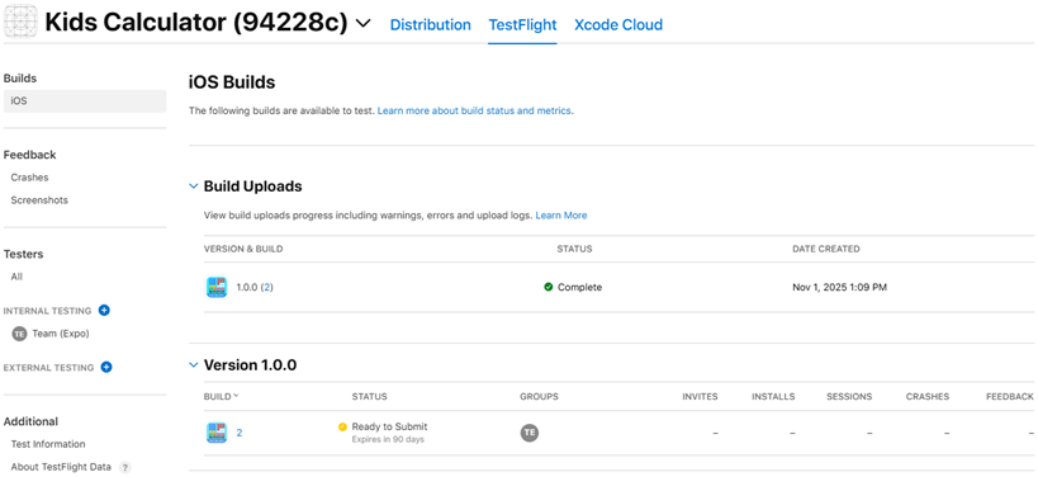

✓ Scheduled iOS submission
Submission details: https://expo.dev/accounts/walamandan/projects/kids-calculator/submissions/185508c7-de6f-47c2-a138-f94eb7e78c2a
Waiting for submission to complete. You can press Ctrl+C to exit.
✓ Submitted your app to Apple App Store Connect!
Your binary has been successfully uploaded to App Store Connect!
- It is now being processed by Apple - you will receive an email when the processing finishes.
- It usually takes about 5-10 minutes depending on how busy Apple servers are.
- When it's done, you can see your build here: https://appstoreconnect.apple.com/apps/6754787401/testflight/ios
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>

Now it will appear in App Store Connect ready for you to fill in the info.

Generate the API key

Copy the ASC App ID information and Build to your word document

If you made it here then it was successful, click and follow the link.



React Native, Expo, and Cross-Platform Publishing

One important advantage of using Visual Studio Code, React Native, and Expo together is that they allow developers to build one project that can run on both Android and iOS. React Native uses JavaScript or TypeScript to create native app interfaces, and Expo adds a toolchain for testing, configuring, building, and submitting the app. During development, Expo Go makes it possible to preview the same project on an iPhone and an Android phone, while Expo Application Services can later create the Android build file and the iOS build file needed for store submission. This does not mean every app is automatically finished for both stores, but it does mean much of the interface, logic, and project structure can be shared instead of written twice.

Some developers choose to build separate native apps instead. In that case, the Android version is commonly written in Kotlin or Java using Android Studio, while the iOS version is commonly written in Swift using Xcode, often with SwiftUI or UIKit for the interface. This approach gives developers direct access to each platform's native tools and design patterns, but it also usually means maintaining two separate codebases, testing two separate apps, and preparing separate store packages for Google Play and the Apple App Store. Other cross-platform options also exist, such as Flutter, which uses the Dart programming language, and .NET MAUI, which uses C# and .NET. The main decision for developers is whether they want one shared project that reaches multiple platforms more efficiently, or separate native projects that give more direct platform-specific control.

Chapter 12 Build a Mobile App with VS Code and Expo Rubric

This chapter focused on creating a mobile app with VS Code and Expo, preparing app settings and assets, testing on a device, checking dependencies, and documenting build readiness.

Category	Points	Evidence of Completion
Project Setup	5	An Expo mobile app project is created, clearly named, opened in VS Code, and organized with the required files and folders.
App Build	5	The app includes the required screens, interface elements, buttons, navigation, or features needed for the project goal.
App Configuration and Assets	5	App settings are checked, including app name, icon, splash screen, version information, package name or bundle identifier, and required assets.
Testing and Revision	5	The app is tested in Expo Go, a simulator, emulator, or on a real device. Layout issues, broken buttons, navigation problems, or errors are identified and revised.
Build Readiness	5	Dependencies, unused files, permissions, screenshots, and basic smoke-test items are checked before building or publishing.
Reflection	5	A short explanation describes the mobile app, the testing process, the build-readiness checks, and relevant computer science vocabulary such as Expo, React Native, dependency, asset, package, permission, version, build, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
Expo mobile app project created	☐
Project opened and organized in VS Code	☐
Required screens or features completed	☐
App name, icon, splash screen, and version checked	☐
Package name or bundle identifier checked	☐
App tested in Expo Go, simulator, emulator, or real device	☐
At least one error, layout issue, or navigation problem revised	☐
Dependencies and permissions checked	☐
Screenshots or project proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the mobile app created with VS Code and Expo. Explain one app setting, asset, or build-readiness item that was checked before publishing. Identify one error, layout issue, navigation problem, or testing result that led to an improvement before final submission.*

13
Publish App on Google Play

This lesson explains the end-to-end process of preparing an Android app for release in the Google Play ecosystem. You will generate the correct Android build file, install and test the app on a real device, and set up a new app record in Google Play Console with the required policy and listing information. You will then walk through internal testing, create release notes, and assemble the store assets that users will see before download, including an app icon, feature graphic, and screenshots from phone and tablet emulators.

Lesson Goal

Students will learn how to test, install, and publish their Android app to the Google Play Store.

Build the Android App

In VS Code terminal:

Command:

eas build -p android --profile production

When the build finishes you'll see a link like: https://expo.dev/artifacts/eas/yourfilename.aab

• AAB = required for the Play Store

• APK = easier for direct testing

To test, build an APK:

Command:

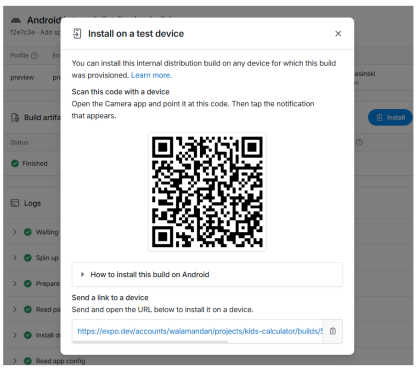
eas build -p android --profile preview

```
🤖 Open this link on your Android devices (or scan the QR code) to install the app:
https://expo.dev/accounts/walamandan/projects/kids-calculator/builds/512f792d-2c47-40a2-a0c5-e6a46f996733
```

Install and Test the APK on Your Phone

Option A: Direct Download

- Click the APK link in your terminal.
- Open the Expo build link on your Android phone.


Install on a test device
You can install this internal distribution build on any device for which this build was provisioned. Learn more.
Scan this code with a device
Open the Camera app and point it at this code. Then tap the notification that appears.
How to install this build on Android
Send a link to a device
Send and open the URL below to install it on a device.

- Tap Download APK.
- When the file finishes downloading, tap it → Install.
- Allow install from unknown apps when prompted.

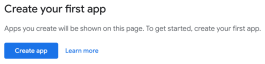
Option B: USB Transfer

- Download the APK onto your computer.
- Connect your phone with a USB cable.
- Copy the APK into your phone's Downloads folder.
- On your phone → open Files → Downloads → tap the file → Install.

Goal: Make sure the app opens correctly, all buttons work, and no red error screens appear.

Create the Google Play App Record

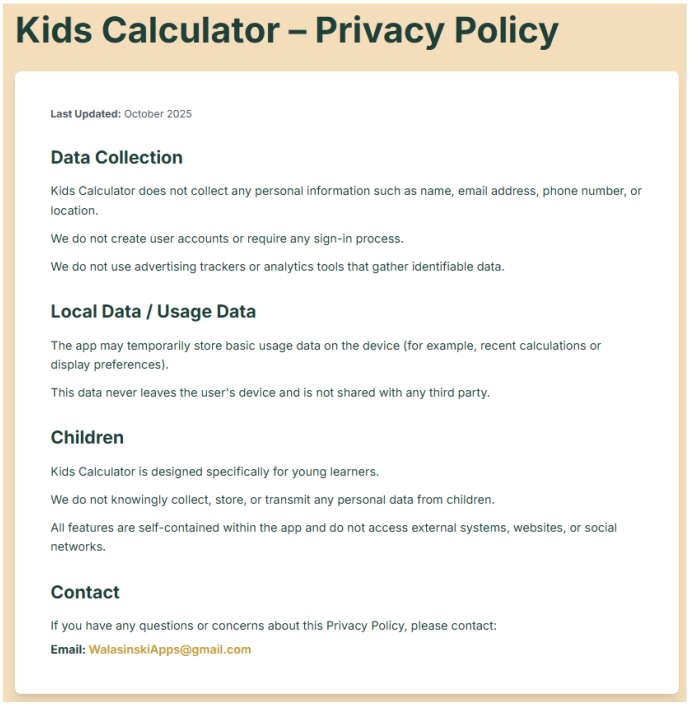
- Go to https://play.google.com/console.
- Click Create App.


Create your first app
Apps you create will be shown on this page. To get started, create your first app.
Create app
Learn more

Fill in details:

- App name, search the google play store for your own app name as well as competitors to help you determine how to name it.
- Don’t select testers yet, complete the Finish setting up your app section next.

Enter your privacy policy URL, a Github page is fine or create your own website and host it there. Here’s mine: I simply had AI generate mine, however I can update it later as needed.



Kids Calculator – Privacy Policy

Last Updated: October 2025

Data Collection

Kids Calculator does not collect any personal information such as name, email address, phone number, or location.

We do not create user accounts or require any sign-in process.

We do not use advertising trackers or analytics tools that gather identifiable data.

Local Data / Usage Data

The app may temporarily store basic usage data on the device (for example, recent calculations or display preferences).

This data never leaves the user's device and is not shared with any third party.

Children

Kids Calculator is designed specifically for young learners.

We do not knowingly collect, store, or transmit any personal data from children.

All features are self-contained within the app and do not access external systems, websites, or social networks.

Contact

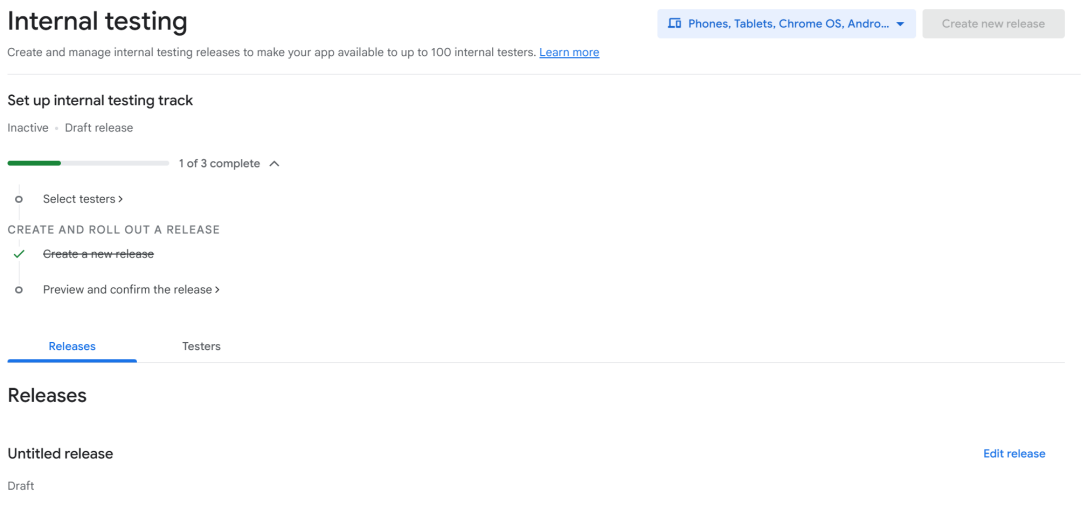
If you have any questions or concerns about this Privacy Policy, please contact:

Email: WalasinskiApps@gmail.com

Complete the rest of the tasks in the dashboard then go back to testing.

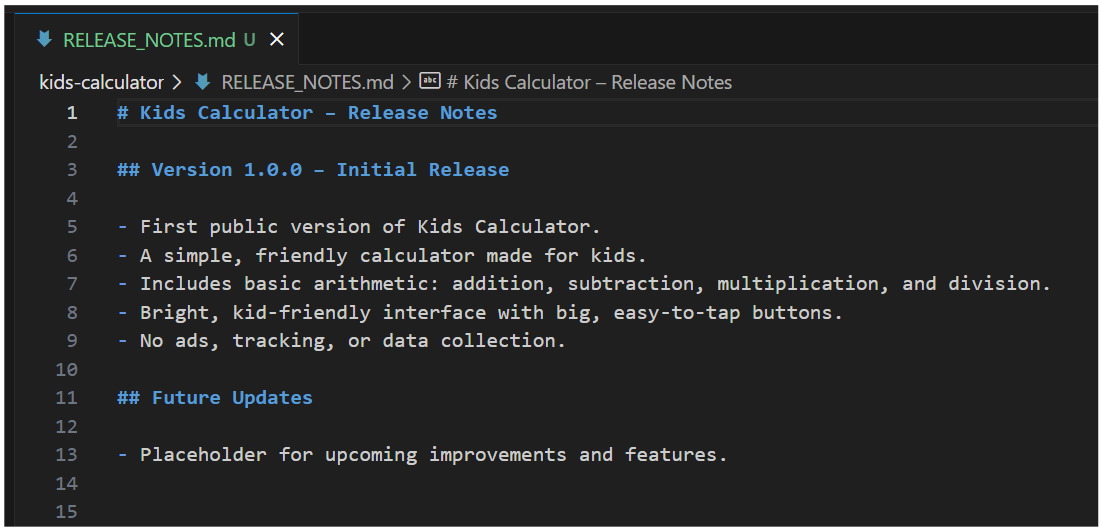
Begin Internal Testing

Choose Edit Release and upload your app package.



Release Notes

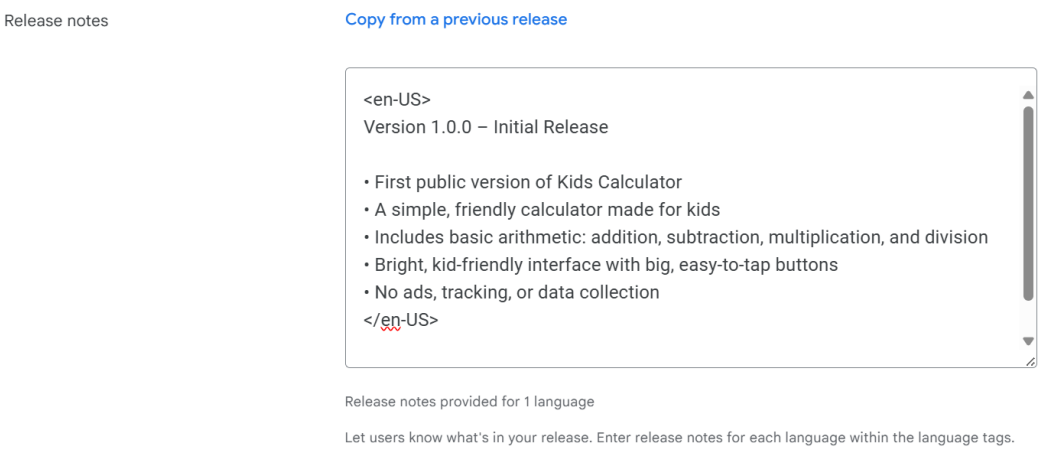
Go inside of VS code and paste this prompt to your AI agent.


RELEASE_NOTES.md U
kids-calculator > RELEASE_NOTES.md > # Kids Calculator – Release Notes
1 # Kids Calculator - Release Notes
2
3 ## Version 1.0.0 - Initial Release
4
5 - First public version of Kids Calculator.
6 - A simple, friendly calculator made for kids.
7 - Includes basic arithmetic: addition, subtraction, multiplication, and division.
8 - Bright, kid-friendly interface with big, easy-to-tap buttons.
9 - No ads, tracking, or data collection.
10
11 ## Future Updates
12
13 - Placeholder for upcoming improvements and features.
14
15

Prompt:

```
Create a new Markdown file named RELEASE_NOTES.md inside my project root
(kids-calculator/).Content requirements:1. Title line: “# Kids Calculator -
Release Notes”2. Add a section for Version 1.0.0 - Initial Release.3. Write 5
short, friendly bullet points describing the app:   - It’s the first public
version.   - It’s a simple calculator made for kids.   - It includes basic
arithmetic (add, subtract, multiply, divide).   - It has a bright, kid-friendly
interface.   - It has no ads or data collection.4. At the end of the file,
include a “## Future Updates” placeholder section so I can add notes later.5.
Use clean Markdown formatting with headings and bullet points.6. When finished,
show me the full Markdown so I can copy it into Google Play or App Store release
notes.
```

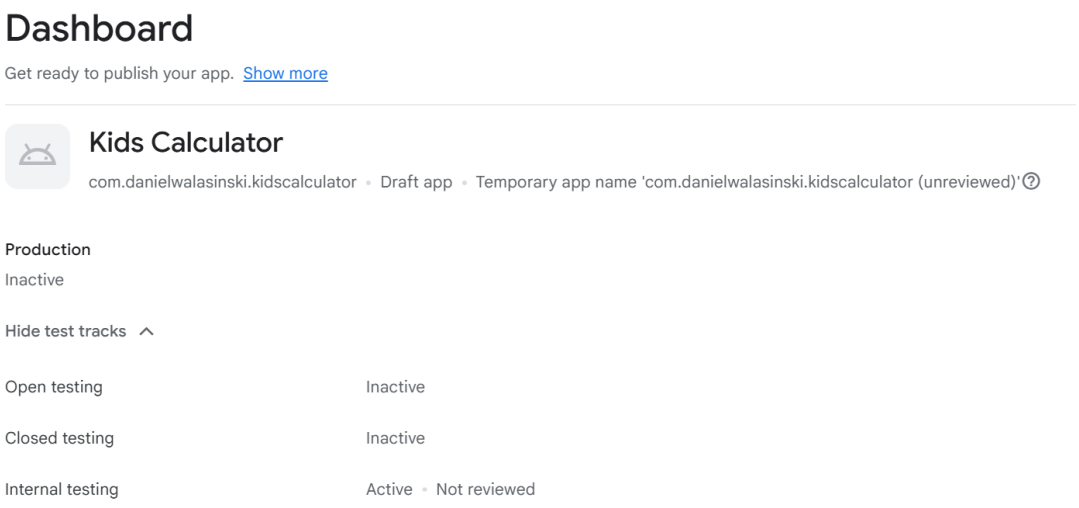
Now copy the output lines 1 to 9 and paste inside your external AI app to reformat from markdown to plain text, then paste into Google Play.


Release notes
Copy from a previous release
<en-US>
Version 1.0.0 – Initial Release
• First public version of Kids Calculator
• A simple, friendly calculator made for kids
• Includes basic arithmetic: addition, subtraction, multiplication, and division
• Bright, kid-friendly interface with big, easy-to-tap buttons
• No ads, tracking, or data collection
</en-US>
Release notes provided for 1 language
Let users know what's in your release. Enter release notes for each language within the language tags.

Finally save and publish

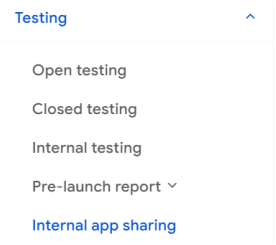
Setup internal testing

- Name the list something like internal testing 1
- Add e-mail address you own and friends, and family, etc with a comma between each one. press enter.
- Copy the web link and save it to your phone so you can let people test it you meet.
- Save changes

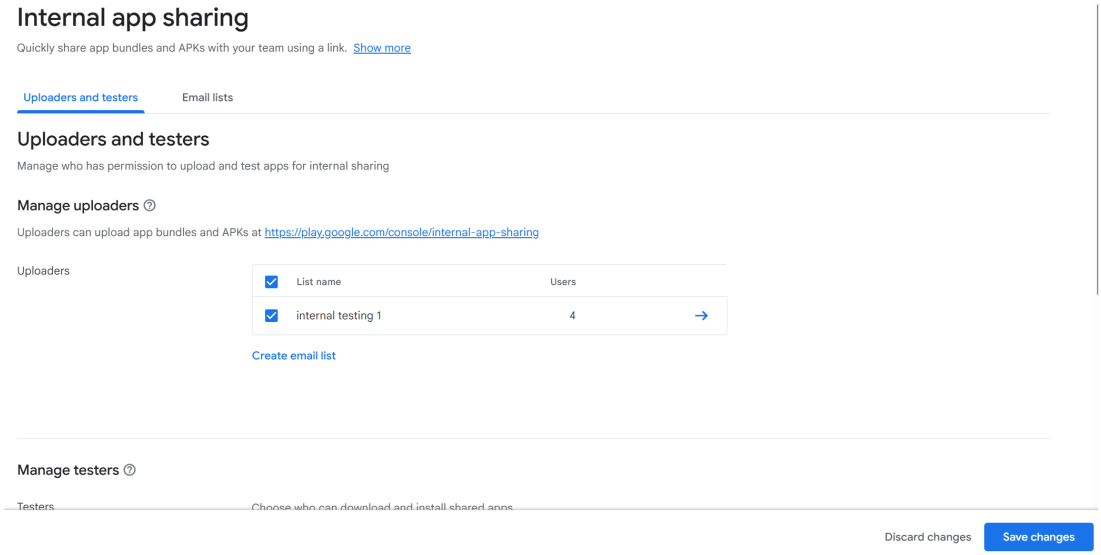


Go to your dashboard, it may look like this.

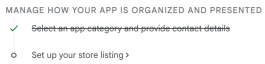
Agree to Terms and Conditions inside of Testing, Internal app sharing.



Select the list and save changes.

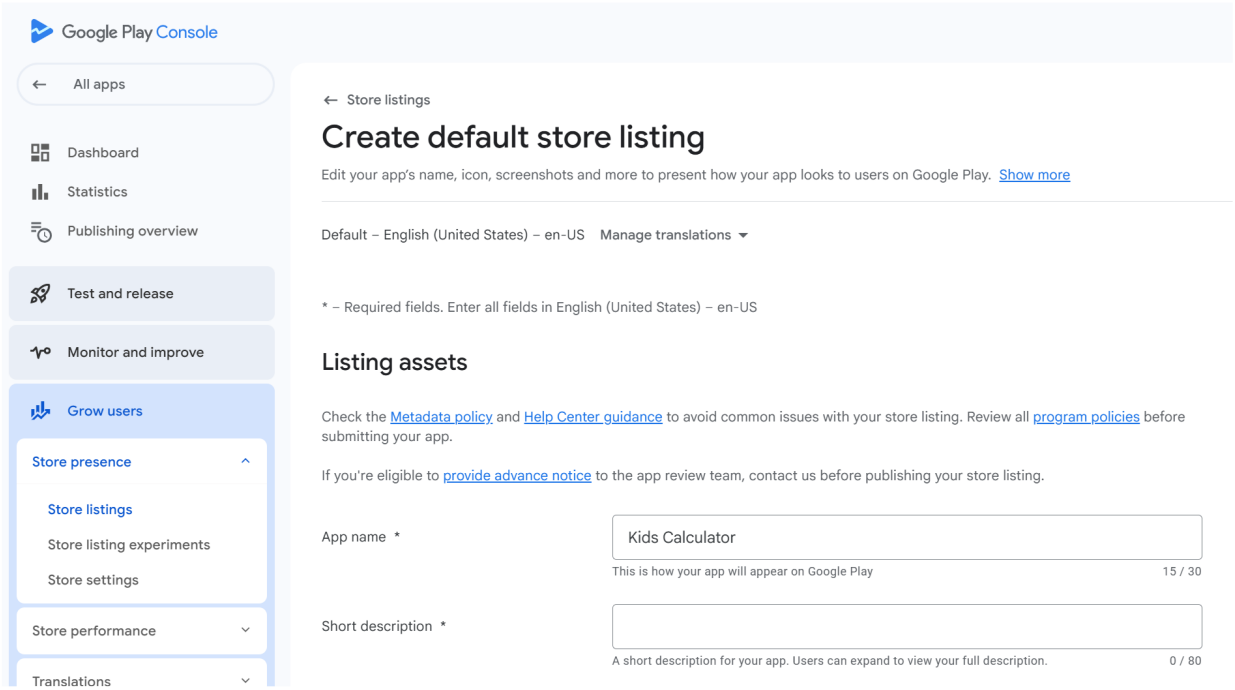


- On the same page enable internal app sharing and save.
- Send the links to the e-mail addresses if they didn't get them automatically and test if it works on all android devices.
- Go back to the dashboard and set up your store listing

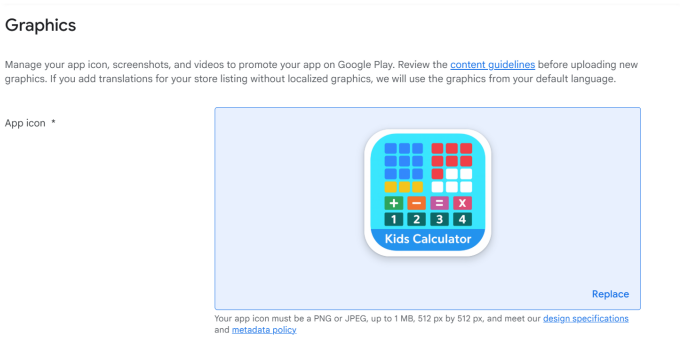

MANAGE HOW YOUR APP IS ORGANIZED AND PRESENTED
Select an app category and provide contact details
Set up your store listing

Google Play Console setup

Click learn more and it will take you to the Google Play Console an external site, complete the setup there.

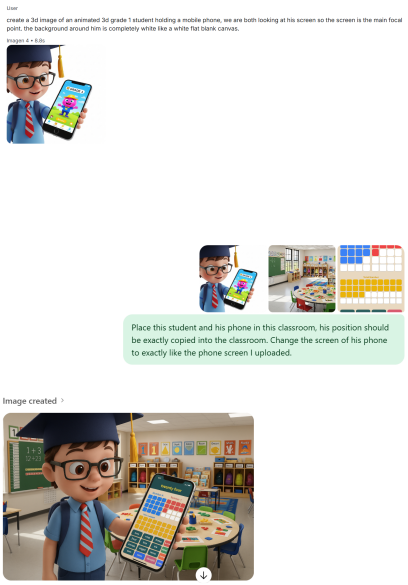

Google Play Console
All apps
Dashboard
Statistics
Publishing overview
Test and release
Monitor and improve
Grow users
Store presence
Store listings
Store listing experiments
Store settings
Store performance
Translations
Store listings
Create default store listing
Edit your app's name, icon, screenshots and more to present how your app looks to users on Google Play. Show more
Default – English (United States) – en-US
Manage translations
* – Required fields. Enter all fields in English (United States) – en-US
Listing assets
Check the Metadata policy and Help Center guidance to avoid common issues with your store listing. Review all program policies before submitting your app.
If you're eligible to provide advance notice to the app review team, contact us before publishing your store listing.
App name *
Kids Calculator
This is how your app will appear on Google Play
15 / 30
Short description *
A short description for your app. Users can expand to view your full description.
0 / 80

Add the App Icon, if you felt like the icon appeared too small from your App packages, this would be a good time to adjust it larger.


Graphics
Manage your app icon, screenshots, and videos to promote your app on Google Play. Review the content guidelines before uploading new graphics. If you add translations for your store listing without localized graphics, we will use the graphics from your default language.
App icon *
Kids Calculator
Replace
Your app icon must be a PNG or JPEG, up to 1 MB, 512 px by 512 px, and meet our design specifications and metadata policy

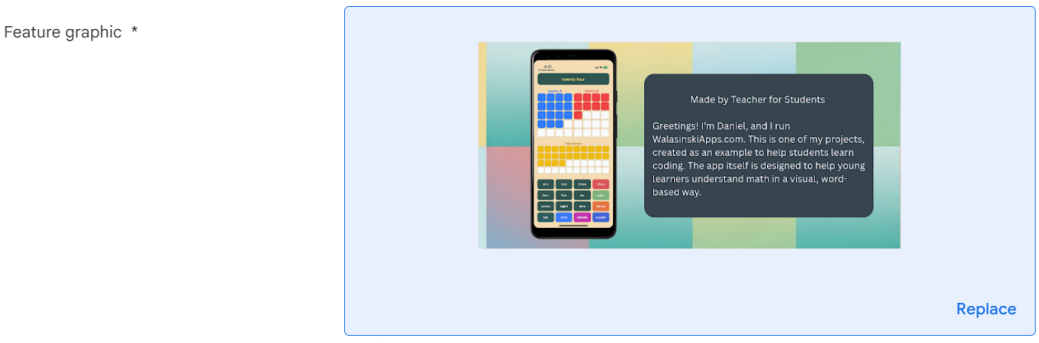
There's the adjustment I made.

Add feature graphic; requirements PNG or JPEG, up to 15 MB, and 1,024 px by 500 px. Create a new design on Canva or Figma to get started. (create new custom size canvas). It will be wide and displayed below the App Icon when users view your app before download.


User
create a 3d image of an animated 3d grade 1 student holding a mobile phone, we are both looking at his screen so the screen is the main focal point. the background around him is completely white like a white flat blank canvas.
Imagen 4 • 8.8s
Place this student and his phone in this classroom, his position should be exactly copied into the classroom. Change the screen of his phone to exactly like the phone screen I uploaded.
Image created

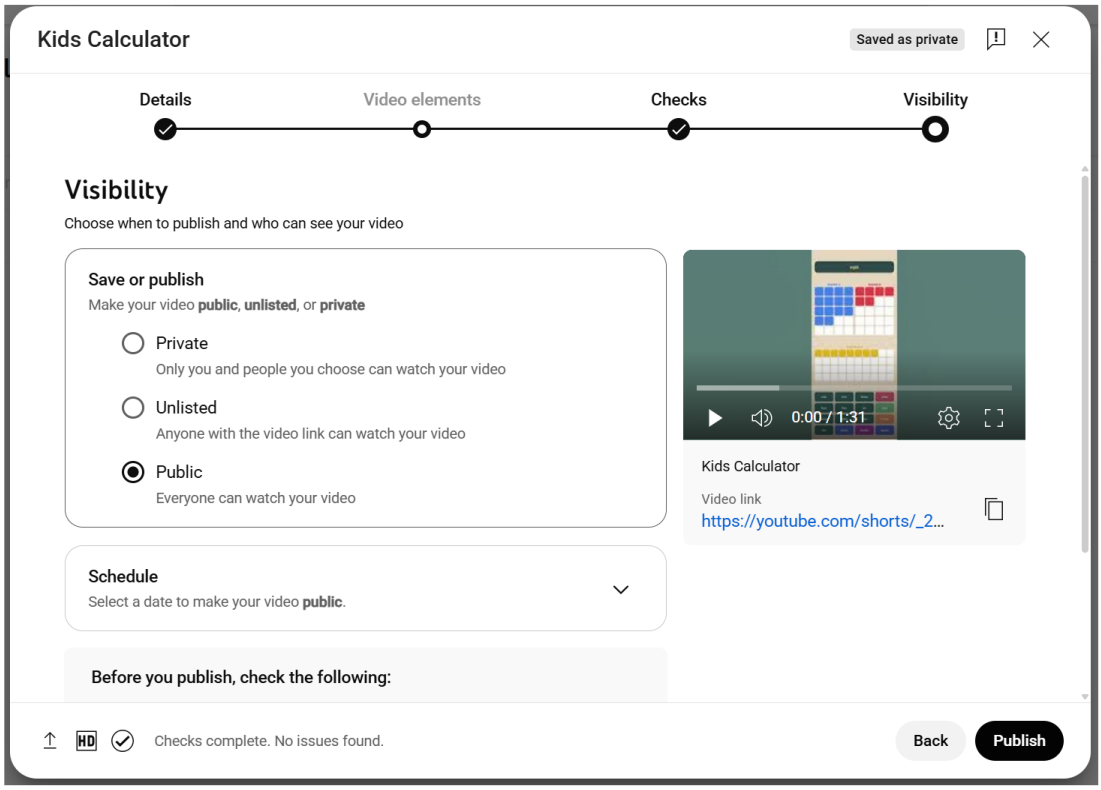
I did some AI prompting to get to this, but you can search for other ideas or go with something super simple like a text banner if you like or a graphic features list, etc. Search the Google Play store apps for more ideas.

However, there were errors with the spelling of the text on the phone so I couldn't use it.


Feature graphic *
Made by Teacher for Students
Greetings! I'm Daniel, and I run WalasinskiApps.com. This is one of my projects, created as an example to help students learn coding. The app itself is designed to help young learners understand math in a visual, word-based way.
Replace
Your feature graphic must be a PNG or JPEG, up to 15 MB, and 1,024 px by 500 px

Ended up using this one for now.

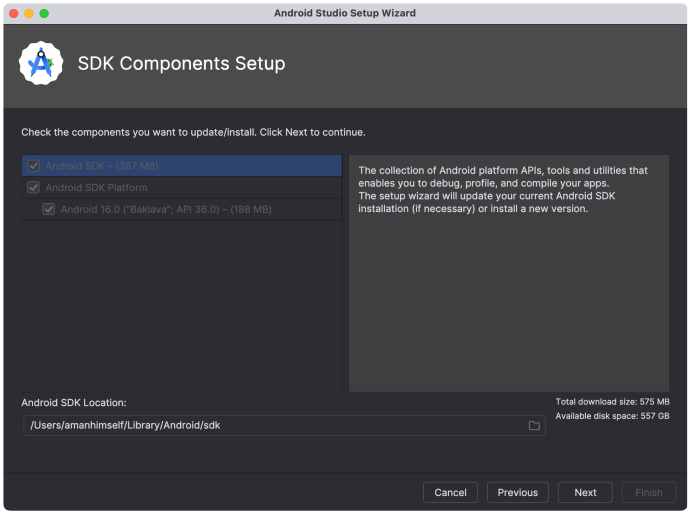
Periodically save as draft. For the video part, you'll scroll from the top of your android phone two times to bring up a menu, you'll see the three mini dots, swipe right one time to find the screen record option. Create the screen recording, make it between 30 seconds to 1 minute and 40 seconds.



Publish it, copy and paste the link into Google Play Console.

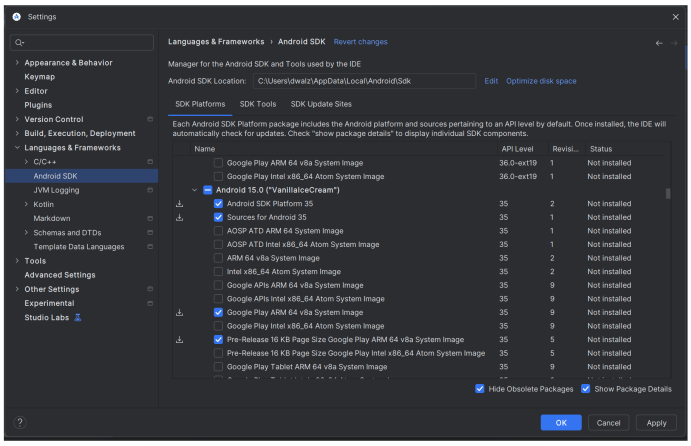
App Screenshots

Install Android Studio and the SDK and Android SDK Platform

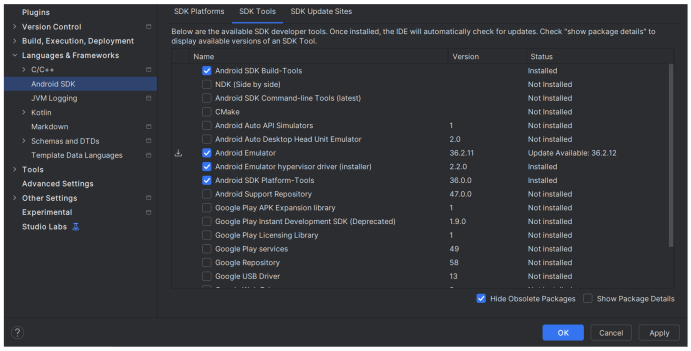

Android Studio Setup Wizard
SDK Components Setup
Check the components you want to update/install. Click Next to continue.
Android SDK – (387 MB)
Android SDK Platform
Android 16.0 ("Baklava"; API 36.0) – (188 MB)
The collection of Android platform APIs, tools and utilities that enables you to debug, profile, and compile your apps.
The setup wizard will update your current Android SDK installation (if necessary) or install a new version.
Android SDK Location:
/Users/amanhimself/Library/Android/sdk
Total download size: 575 MB
Available disk space: 557 GB
Cancel
Previous
Next
Finish

Android Studio will install the latest version of the Android SDK. However, Android 15 (VanillaIceCream) SDK is required to compile a React Native app.

Open Android Studio, go to Settings > Languages & Frameworks > Android SDK. From the SDK Platforms tab first enable Show Package Details, Scroll down to Android 15 VanillaIceCream, select Android SDK Platform 35 and Sources for Android 35 as well as Google Play ARM 64 v8a System Image, and Pre-release 16kb page size Google Play ARM 64 v8a System Image

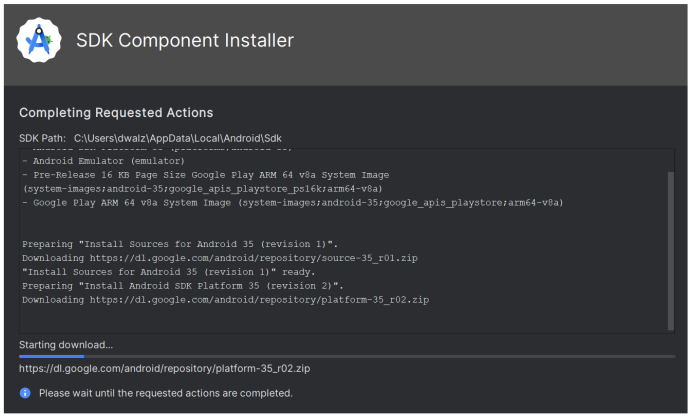

Settings
Languages & Frameworks › Android SDK
Revert changes
Manager for the Android SDK and Tools used by the IDE
Android SDK Location: C:\Users\dwalz\AppData\Local\Android\Sdk
Edit
Optimize disk space
SDK Platforms
SDK Tools
SDK Update Sites
Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, the IDE will automatically check for updates. Check "show package details" to display individual SDK components.
Name
API Level
Revisi...
Status
Google Play ARM 64 v8a System Image
Google Play Intel x86_64 Atom System Image
Android 15.0 ("VanillaIceCream")
Android SDK Platform 35
Sources for Android 35
AOSP ATD ARM 64 System Image
AOSP ATD Intel x86_64 Atom System Image
ARM 64 v8a System Image
Intel x86_64 Atom System Image
Google APIs ARM 64 v8a System Image
Google APIs Intel x86_64 Atom System Image
Google Play ARM 64 v8a System Image
Google Play Intel x86_64 Atom System Image
Pre-Release 16 KB Page Size Google Play ARM 64 v8a System Image
Pre-Release 16 KB Page Size Google Play Intel x86_64 Atom System Image
Google Play Tablet ARM 64 v8a System Image
Hide Obsolete Packages
Show Package Details
OK
Cancel
Apply

Click on the SDK Tools tab and make sure you have at least one version of the Android SDK Build-Tools and Android Emulator installed.



Copy or remember the path listed in the box that says Android SDK Location.

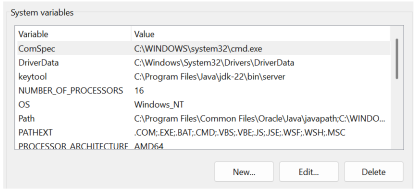
Press OK to begin downloading


SDK Component Installer
Completing Requested Actions
SDK Path: C:\Users\dwalz\AppData\Local\Android\Sdk
- Android Emulator (emulator)
- Pre-Release 16 KB Page Size Google Play ARM 64 v8a System Image
(system-images;android-35;google_apis_playstore_ps16k;arm64-v8a)
- Google Play ARM 64 v8a System Image (system-images;android-35;google_apis_playstore;arm64-v8a)
Preparing "Install Sources for Android 35 (revision 1)".
Downloading https://dl.google.com/android/repository/source-35_r01.zip
"Install Sources for Android 35 (revision 1)" ready.
Preparing "Install Android SDK Platform 35 (revision 2)".
Downloading https://dl.google.com/android/repository/platform-35_r02.zip
Starting download...
https://dl.google.com/android/repository/platform-35_r02.zip
Please wait until the requested actions are completed.

Add environment variables to computer

Press Windows Key, type “Edit environment variables” → Open it.

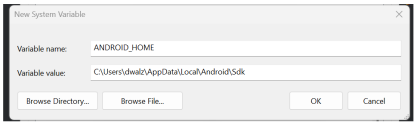
In the System Properties window → Click Environment Variables...



Under User variables, click New...

Variable name: ANDROID_HOME

Variable value: (paste your SDK path from Android Studio)

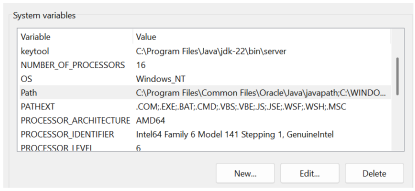


Click OK.

Add platform-tools and emulator to your PATH

Still in Environment Variables:

Find the variable called Path → select → click Edit.



Click New, then add these two lines (replace your username if needed):

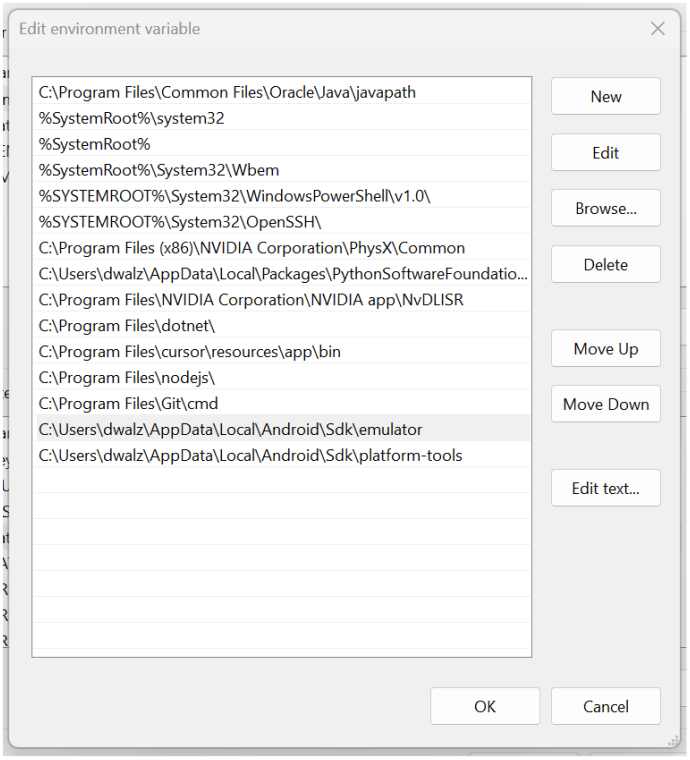
Code:

```
C:\Users\<yourname>\AppData\Local\Android\Sdk\platform-tools
```

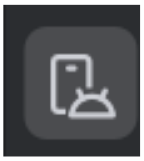
Code:

```
C:\Users\<yourname>\AppData\Local\Android\Sdk\emulator
```

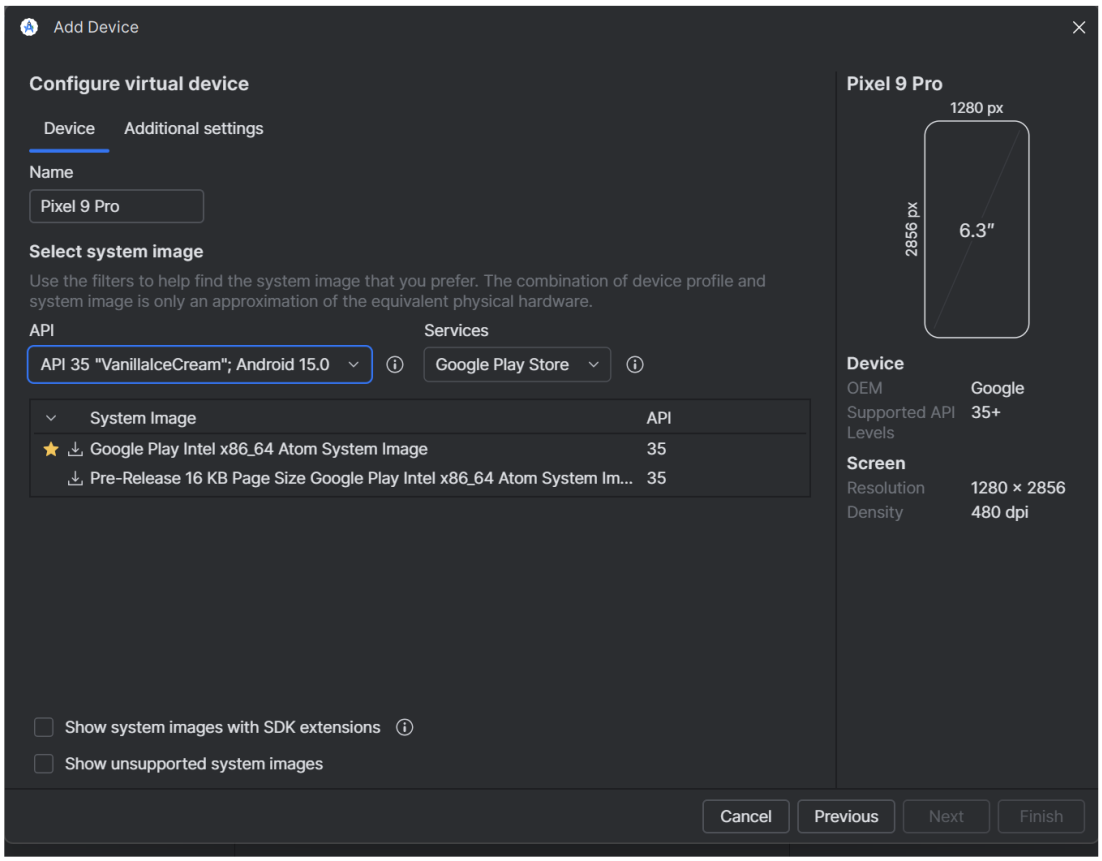
Click ok on the menus to close them



Open Android Studio and on the right panel open Device Manager then click the plus sign and create virtual devices.



Select the Google Pixel 9 Pro and enable the Vanilla Ice Cream API, click the star to download the SDK.



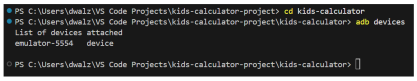
Go back and enable the Medium Tablet and the Pixel Tablet, download the SDKs for them too.

Inside of Android Studio run the Google Pixel 9 Pro Virtual Device by clicking the add button and selecting it.

Open your project terminal in VS code and enter:

Command:

```
adb devices
```


PS C:\Users\dwalz\VS Code Projects\kids-calculator-project> cd kids-calculator
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator> adb devices
List of devices attached
emulator-5554 device
PS C:\Users\dwalz\VS Code Projects\kids-calculator-project\kids-calculator>

The emulator should be listed, if it is not then restart the computer and check the steps above that you have all of the necessary SDKs also Gemini AI is available inside of Android Studio.

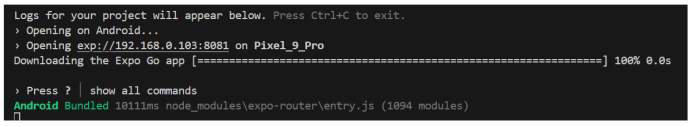
Start Expo's local development server

Command:

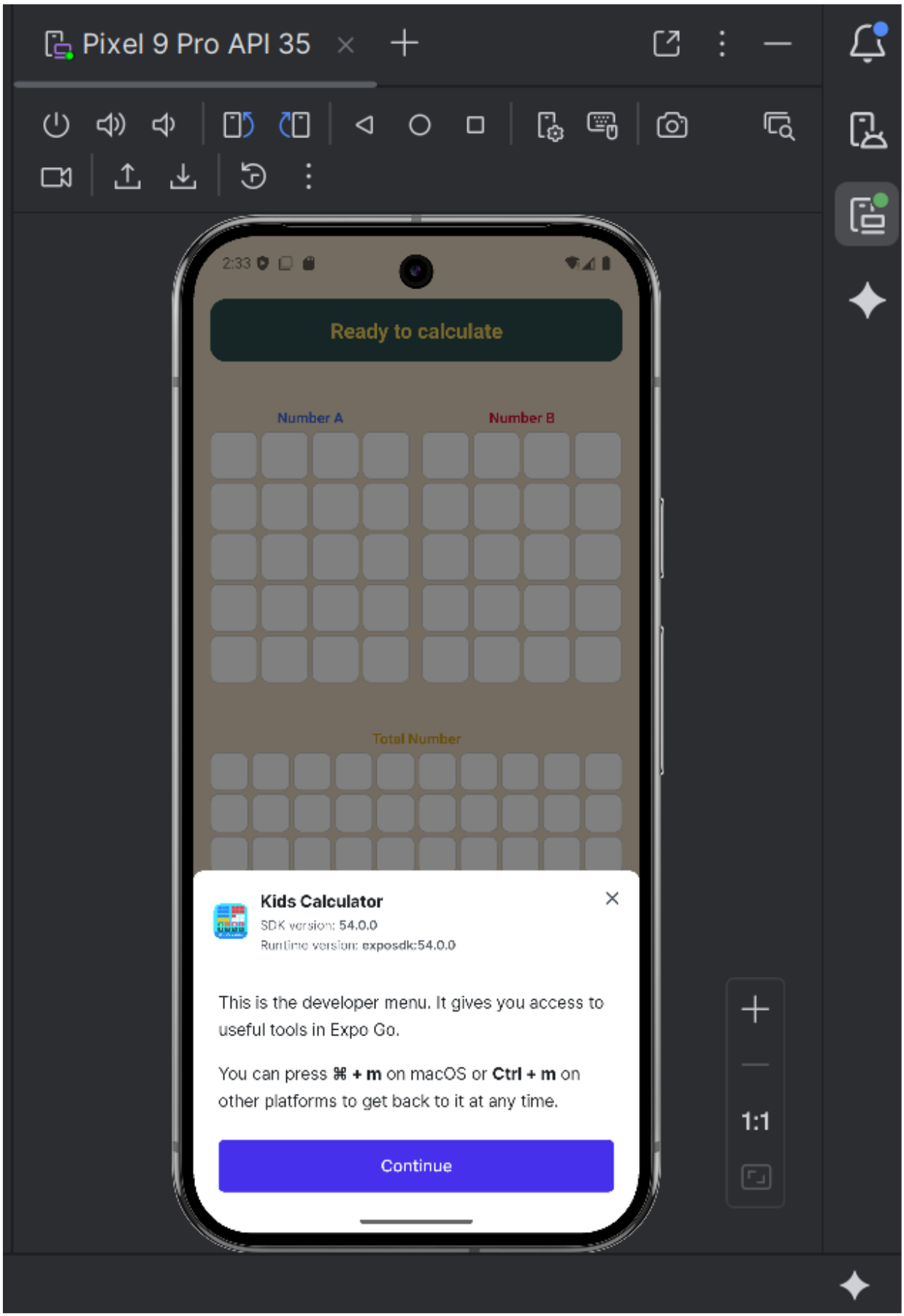
```
npx expo start --clear
```

Launch your app on the emulator by pressing:

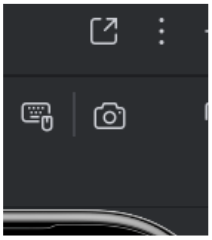
```
A
```

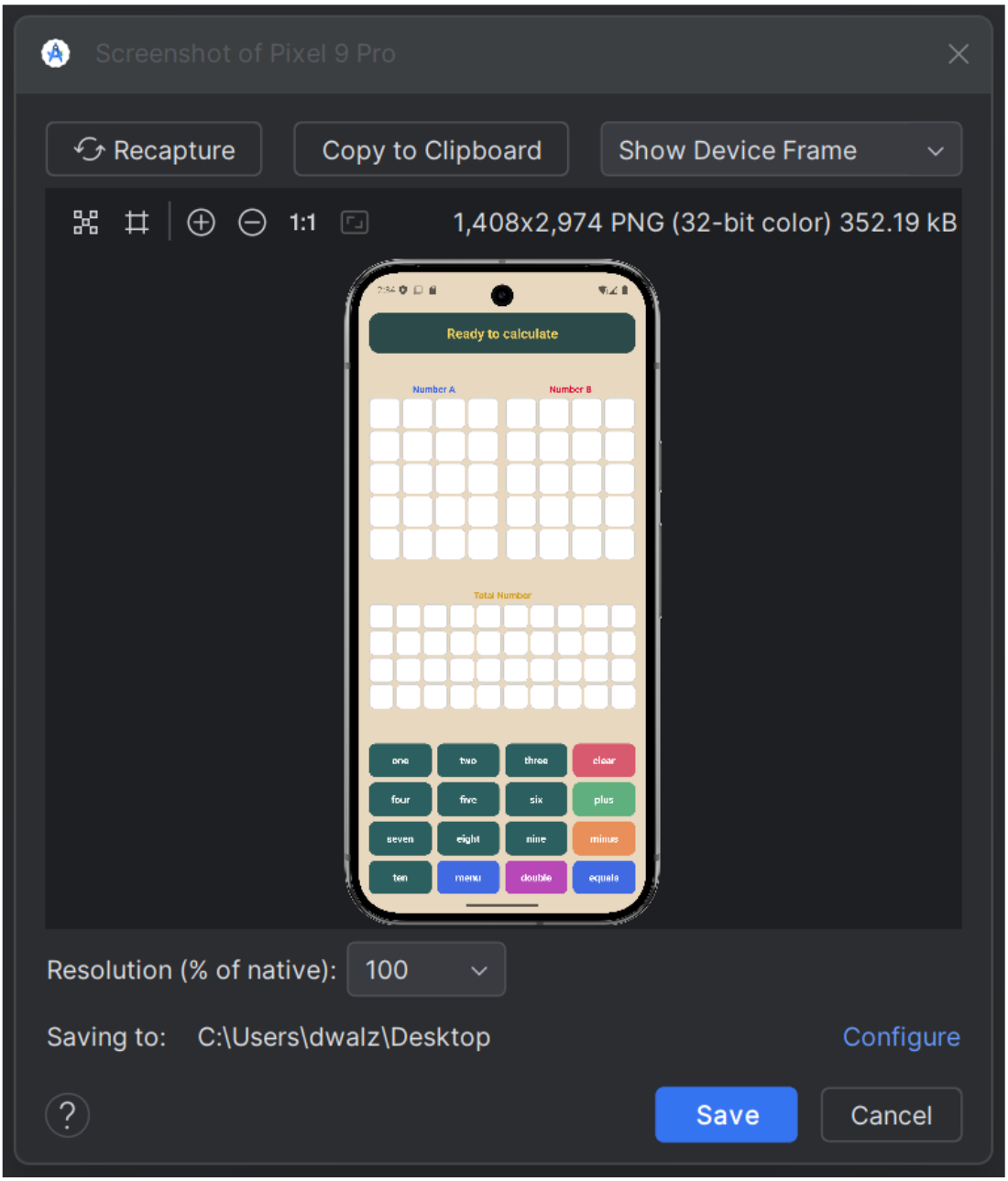

Logs for your project will appear below. Press Ctrl+C to exit.
› Opening on Android...
› Opening exp://192.168.0.103:8081 on Pixel_9_Pro
Downloading the Expo Go app [==========] 100% 0.0s
› Press ? | show all commands
Android Bundled 10111ms node_modules\expo-router\entry.js (1094 modules)

Mine took a long time to download, here's the result inside of Android Studio.

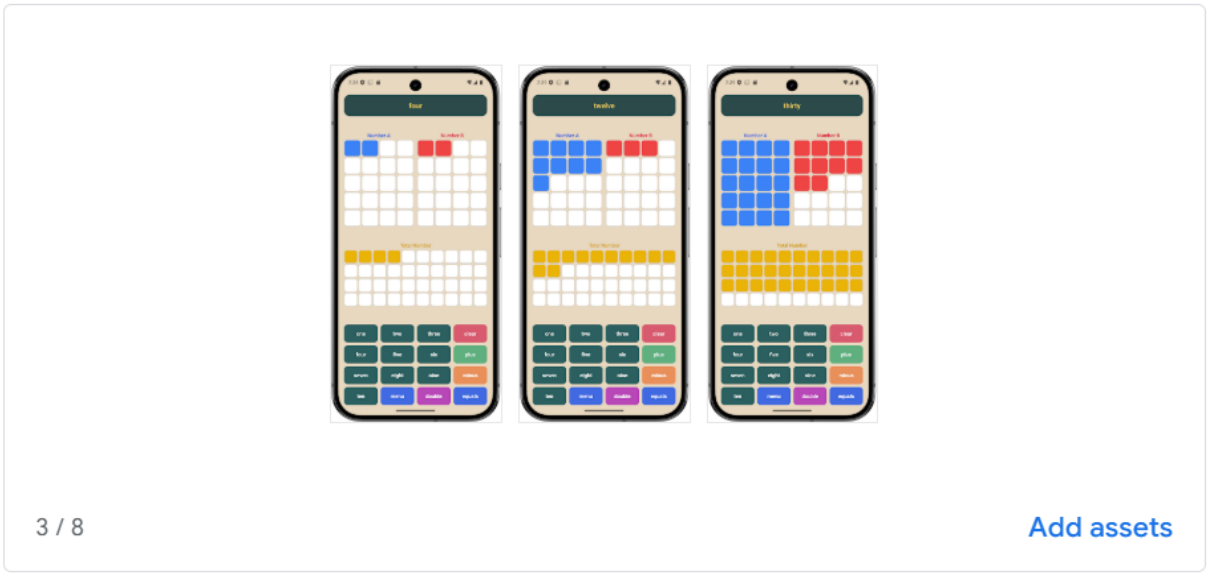

Pixel 9 Pro API 35
2:33
Ready to calculate
Number A
Number B
Total Number
Kids Calculator
SDK version: 54.0.0
Runtime version: exposdk:54.0.0
This is the developer menu. It gives you access to useful tools in Expo Go.
You can press ⌘ + m on macOS or Ctrl + m on other platforms to get back to it at any time.
Continue
1:1

Find the camera button to take screenshots, for Google Play you'll need 2-8. Each screen should display something different if possible. Enable show device frame.



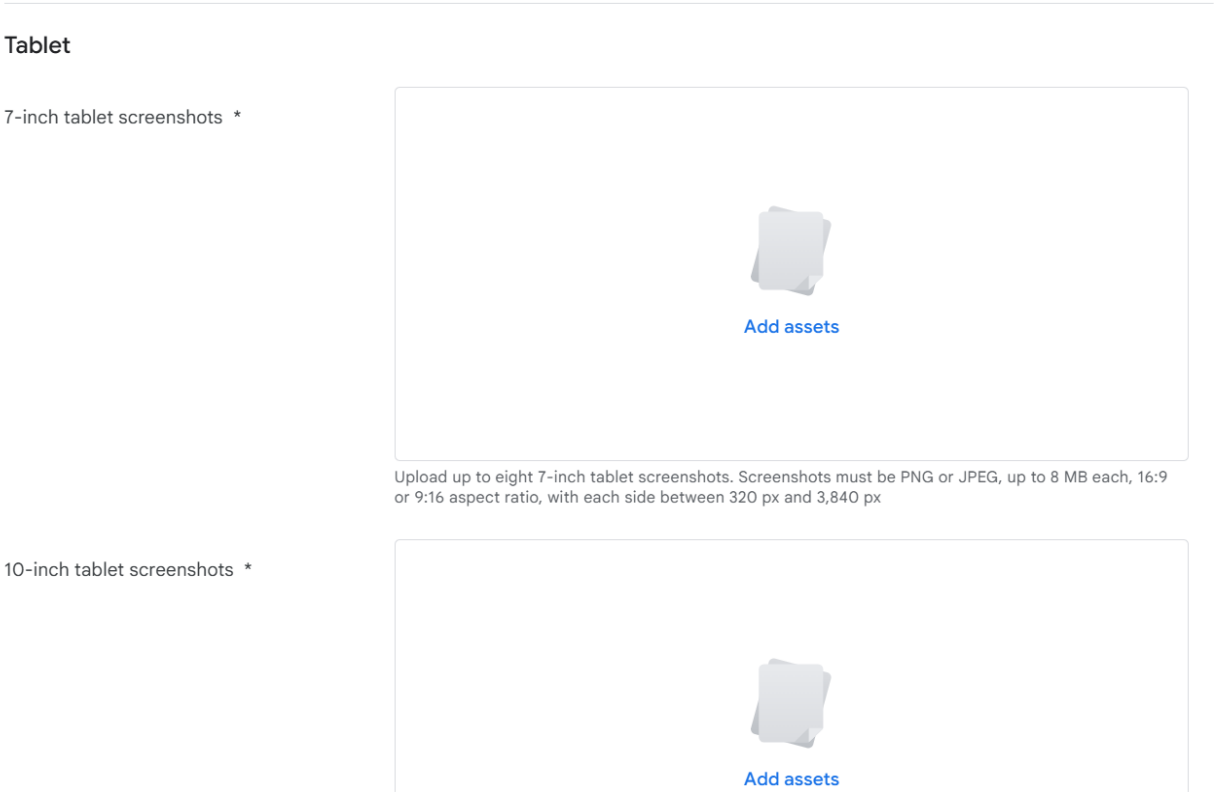


Upload them to in your app project inside of the store listing screenshots area.

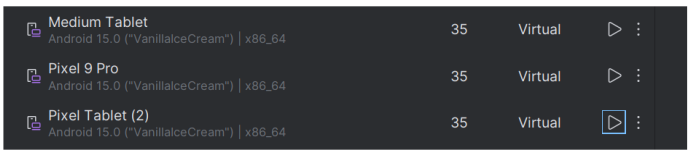


Upload 2-8 phone screenshots. Screenshots must be PNG or JPEG, up to 8 MB each, 16:9 or 9:16 aspect ratio, with each side between 320 px and 3,840 px

If you enabled you app to be used on tablets then you will need to create emulators and stake screenshots of them too.


Tablet
7-inch tablet screenshots *
Add assets
Upload up to eight 7-inch tablet screenshots. Screenshots must be PNG or JPEG, up to 8 MB each, 16:9 or 9:16 aspect ratio, with each side between 320 px and 3,840 px
10-inch tablet screenshots *
Add assets

You should have added the Medium Tablet and the Pixel Tablet earlier

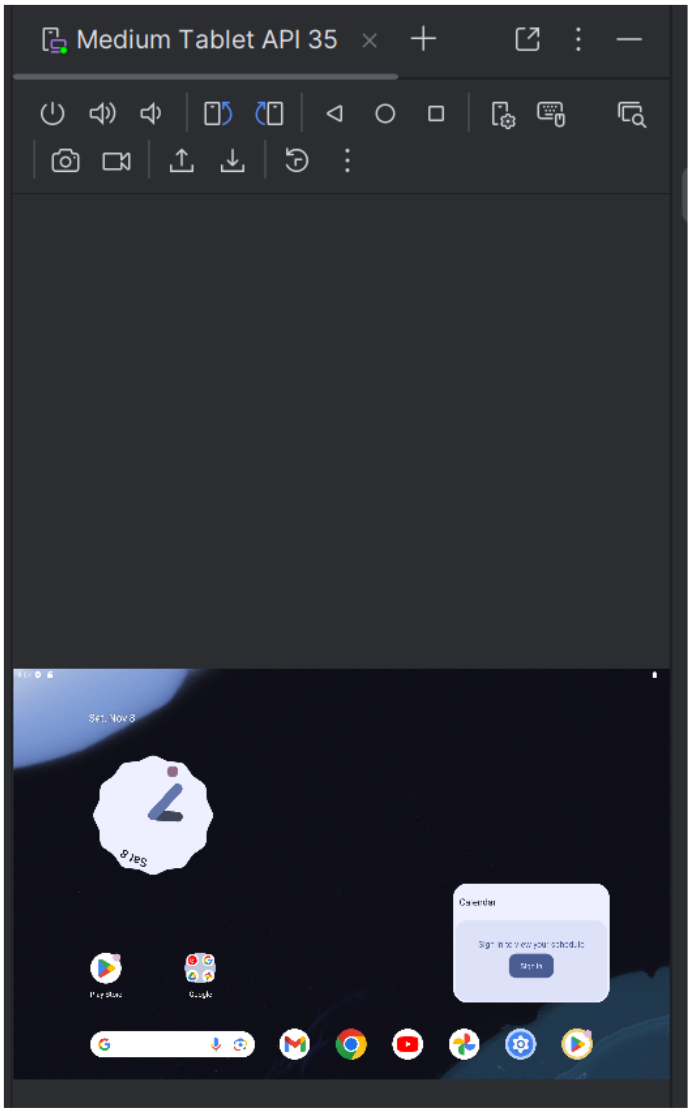


Inside your ExpoVS code project press (a) to open Android or control c to stop the server and restart it by entering

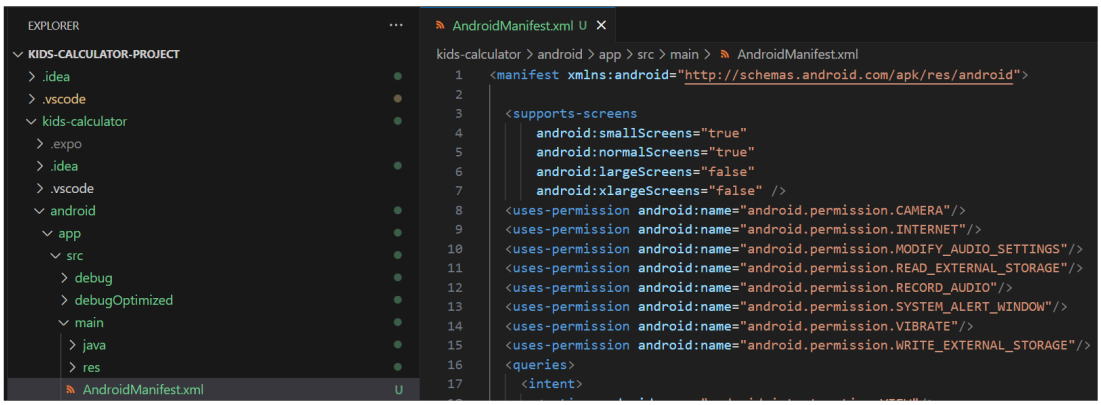
Command:

```
npx expo start --clear
```

Now press a and make sure your android emulator is running the medium tablet as a virtual device.



After you test on tablet and mobile devices you can update the file AndroidManifest.xml file to tell Google Play Console which device types are supported.

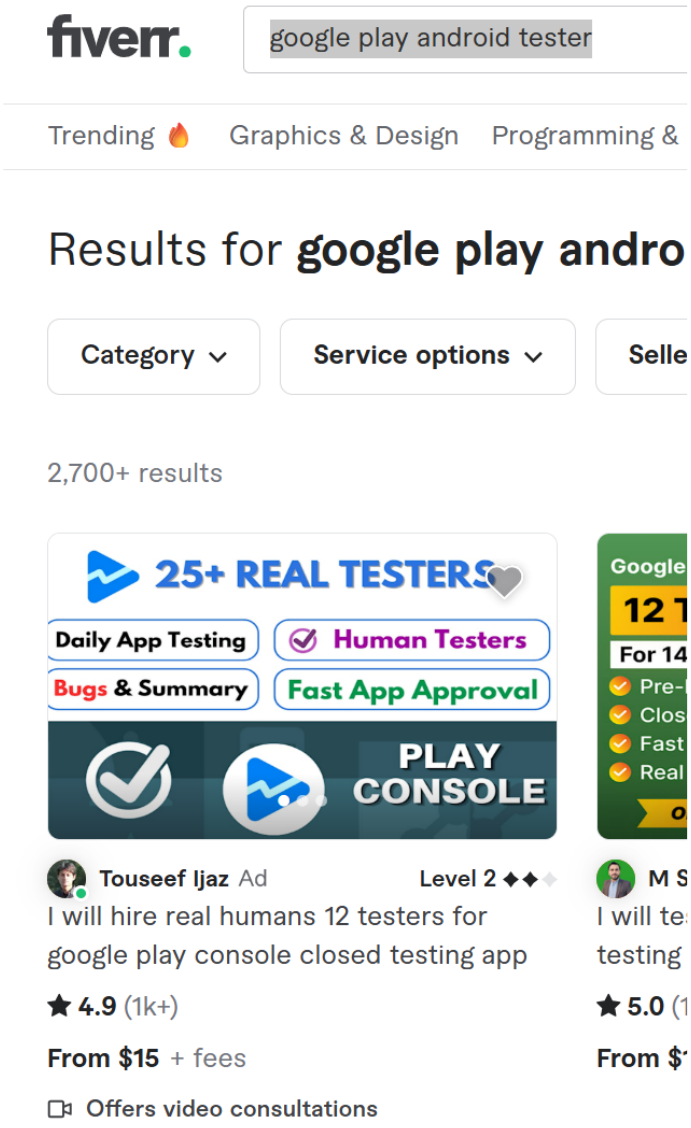


Then publish a new app package file using the code

Command:

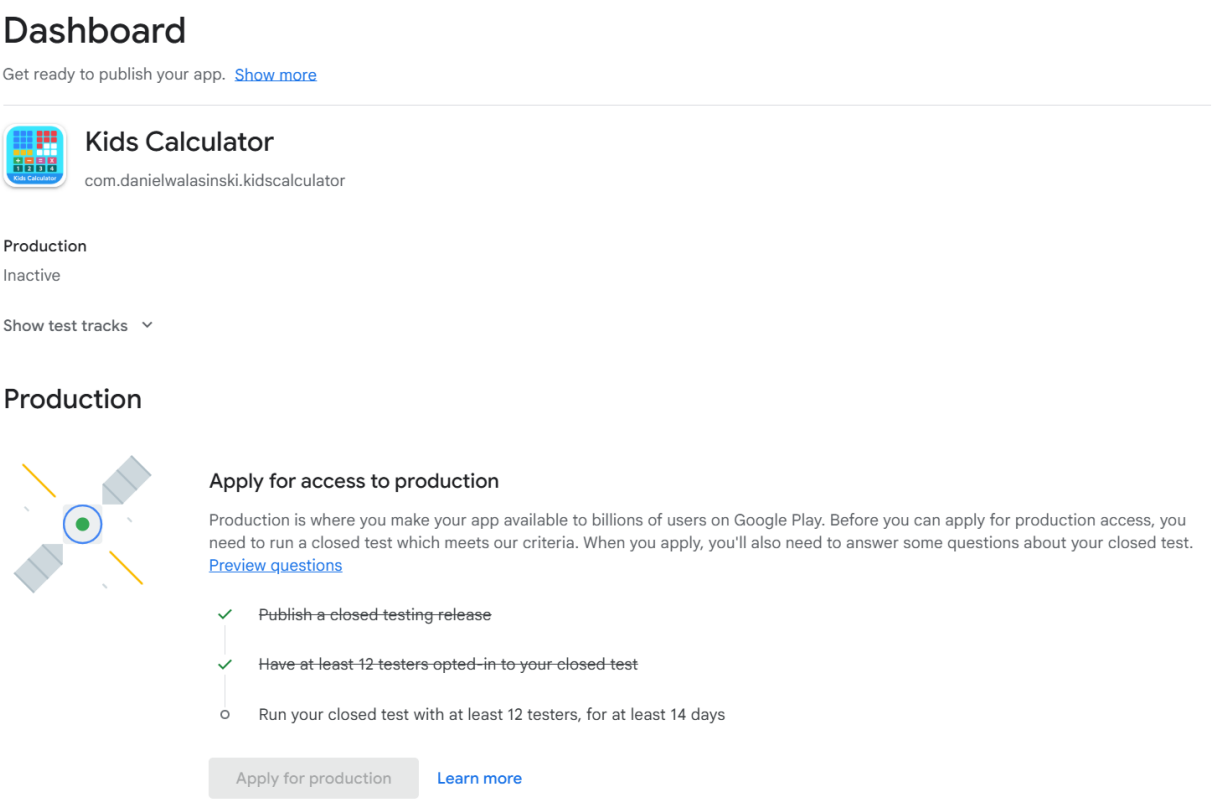
```
npx eas build --platform android --profile production
```

You'll need a group of 20 testers and if you haven't found them, you can hire some quickly on Fiverr by making a simple search: google play android tester, and look for sellers who have the most reviews and read them to see if they sound legitimate. Alternatively, there's a group on Reddit called r/AndroidTesting where it's easy to find others willing to cooperate.



I purchased the service from the first listing, uploaded my new .aab file to Google Play

I downloaded the tester e-mail list and set up closed testing for them first, then after 12 opted in, they will need to complete the closed test over 14 days until you can apply for production.


Dashboard
Get ready to publish your app. Show more
Kids Calculator
com.danielwalasinski.kidscalculator
Production
Inactive
Show test tracks
Production
Apply for access to production
Production is where you make your app available to billions of users on Google Play. Before you can apply for production access, you need to run a closed test which meets our criteria. When you apply, you'll also need to answer some questions about your closed test.
Preview questions
Publish a closed testing release
Have at least 12 testers opted-in to your closed test
Run your closed test with at least 12 testers, for at least 14 days
Apply for production
Learn more

App Store Research and Positioning

For your first published app, try to keep the project simple. Focus on learning the process and the workflow, because it becomes easier each time you develop a new app. Study other apps in the same category. What features do they have? What do their screenshots and descriptions communicate? Do they look like they belong to a large organization or an independent developer? What do their icon, name, colors, and screenshots signal about their target audience and positioning? Is there anything your app does better, or are there features worth highlighting more clearly?

Use this research to improve your store listing, revise your features, or plan your next project. When someone asks you to develop an app, or when you have your own idea, start by working backward. What will the time commitment be? What costs are involved? How large is the project? Which APIs or services will you need? You learned the app development process step by step from scratch.

Now you can begin reverse engineering other apps, identifying a niche, and estimating what it would actually take to compete in a specific area of the app store. Some apps are discovered through store search or browsing, while others grow because core users share them with people who would truly benefit from using them. You can also make a list of the apps you personally use and why you use them. The list may be smaller than you expect. Your phone may contain many apps, but how many of them do you actually open every week?

Chapter 13 Test and Publish a Mobile App on Google Play Rubric

This chapter focused on preparing an Android app for Google Play, creating a build, testing the app, preparing store listing materials, and documenting the publishing process.

Category	Points	Evidence of Completion
Project Preparation	5	The mobile app project is reviewed and prepared for Android publishing, including app name, version, package name, icon, splash screen, and required assets.
Android Build	5	An Android build is created or attempted using the required build process. The build file, build status, or build attempt is documented.
App Testing	5	The app is tested on an Android device, emulator, or approved testing method. Crashes, layout issues, broken buttons, or other problems are identified and revised.
Store Listing Materials	5	Store listing materials are prepared, including app description, screenshots, release notes, app icon, feature graphic, and privacy policy information when required.
Evidence of Work	5	Screenshots, build proof, testing proof, store listing draft, release notes, or other project evidence are submitted.
Reflection	5	A short explanation describes the Android publishing process, the testing results, and relevant computer science vocabulary such as Android, build, package name, version, release notes, store listing, testing, privacy, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
App name and version checked	☐
Android package name checked	☐
App icon and splash screen checked	☐
Android build created or attempted	☐
App tested on Android device, emulator, or approved method	☐
At least one issue or improvement revised	☐
Store description or listing draft prepared	☐
Screenshots or store graphics prepared	☐
Build proof, testing proof, or store listing proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the process used to prepare the app for Google Play. Explain one testing result, build issue, or store listing requirement that was checked. Identify one problem, improvement, or publishing detail that was revised before final submission.*

14
Publish App on Apple App Store

This lesson explains how an iOS app moves from a working Expo project to a release that can be tested on a real iPhone and submitted to the App Store. You will create an Apple Developer account, build an iOS production package using Expo Application Services, and upload the build to App Store Connect. You will then use TestFlight to install the app on an iPhone, confirm that it runs correctly outside the development environment, and verify that the user experience matches what you plan to publish.

Lesson Goal

Students will learn how to test their iOS app on an iPhone using TestFlight and publish it to the App Store.

Apple Developer Account

Create a developer account at https://developer.apple.comit will cost $100 usd per year.

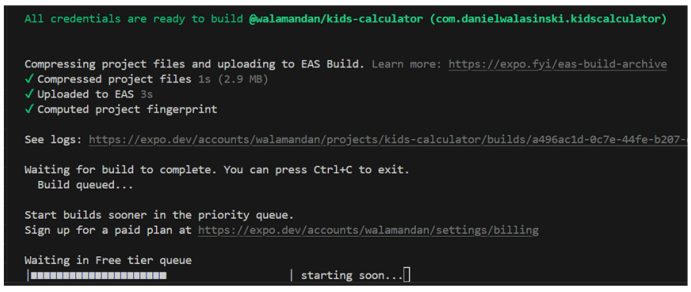
Build the iOS App

In VS Code terminal:

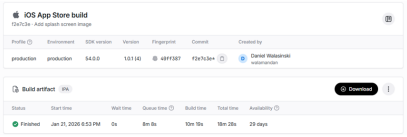
Command:

```
eas build -p ios --profile production
```

The terminal will prompt you to login to your Apple Developer Account using e-mail and password. Next wait awhile while the process completes.


All credentials are ready to build @walamandan/kids-calculator (com.danielwalasinski.kidscalculator)
Compressing project files and uploading to EAS Build. Learn more: https://expo.fyi/eas-build-archive
✓ Compressed project files 1s (2.9 MB)
✓ Uploaded to EAS 3s
✓ Computed project fingerprint
See logs: https://expo.dev/accounts/walamandan/projects/kids-calculator/builds/a496ac1d-0c7e-44fe-b207-
Waiting for build to complete. You can press Ctrl+C to exit.
Build queued...
Start builds sooner in the priority queue.
Sign up for a paid plan at https://expo.dev/accounts/walamandan/settings/billing
Waiting in Free tier queue
| starting soon...

If it doesn't finish by itself you may want to click the link and check for your package

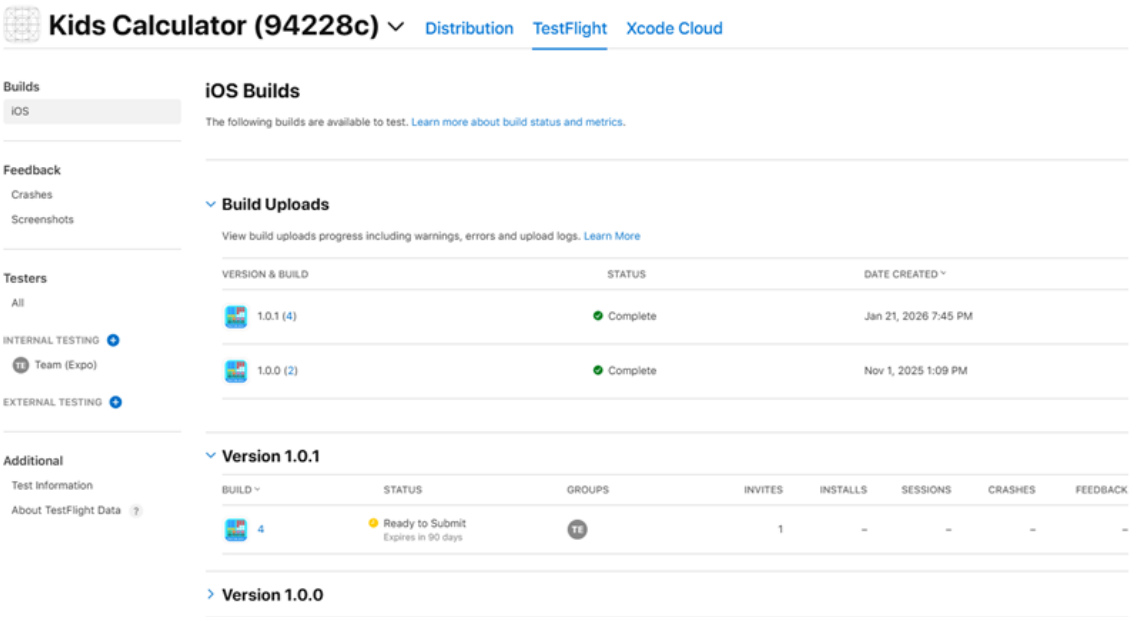


Then from here you can download your app package.

Install and Test on an iPhone

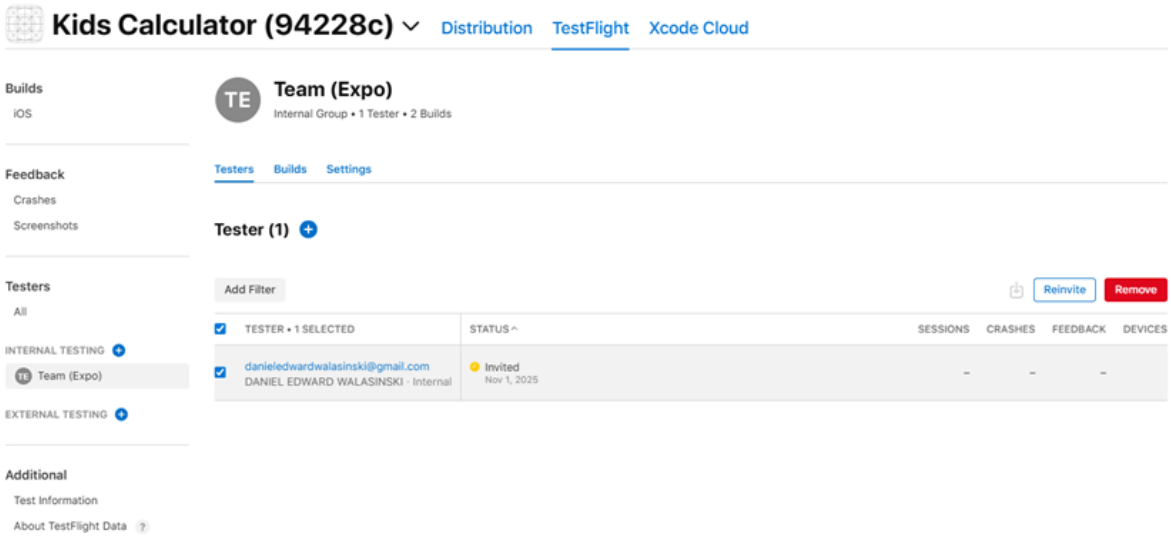
- Inside the VS Code terminal type: eas submit -p ios
- Choose: Select a build from EAS
- Wait for upload to finish.
- After about 10 minutes click the link provided in the terminal to find your app in App Store Connect. https://appstoreconnect.apple.com/apps

Scheduled iOS submission
Submission details:
Waiting for submission to complete. You can press Ctrl+C to exit.
Submitting your app to Apple App Store Connect: waiting for an available submitter

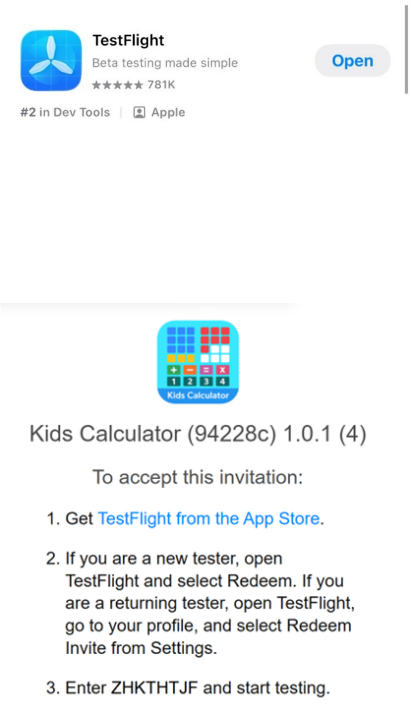


Add yourself as a tester and send the invite to your email and get the code you'll use for Test Flight on your iPhone.

Install TestFlight on your iPhone from the App Store.

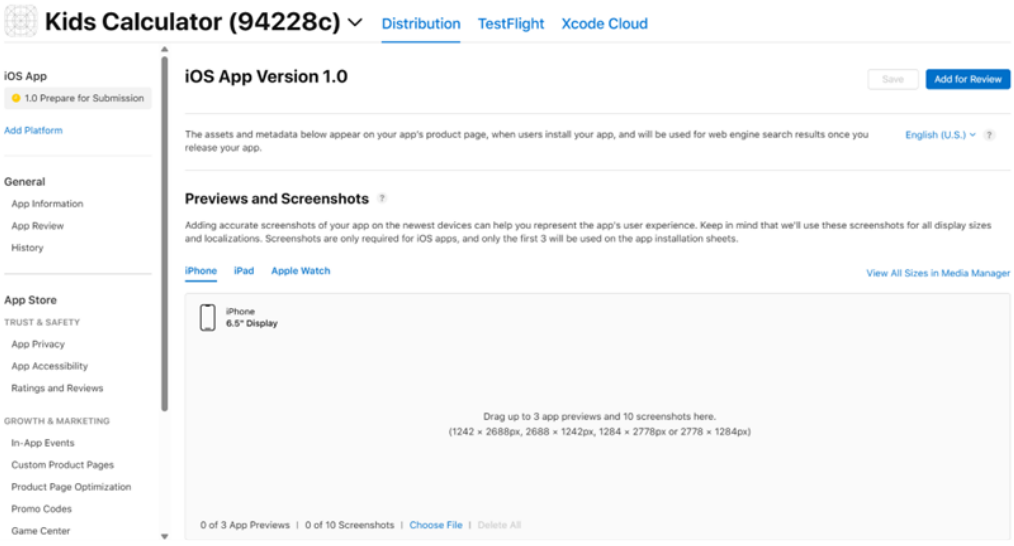


Enter the code from the email in the app. The app will now be playable from your iPhone.


TestFlight
Beta testing made simple
★★★★★ 781K
Open
#2 in Dev Tools
Apple
Kids Calculator
Kids Calculator (94228c) 1.0.1 (4)
To accept this invitation:
1. Get TestFlight from the App Store.
2. If you are a new tester, open TestFlight and select Redeem. If you are a returning tester, open TestFlight, go to your profile, and select Redeem Invite from Settings.
3. Enter ZHKTHTJF and start testing.

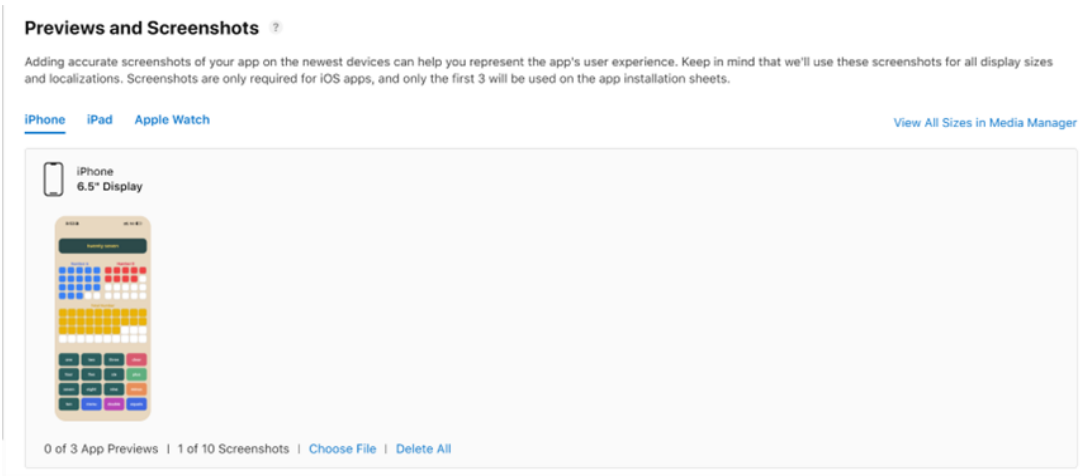
Prepare App For Submission

Add all of the information needed inside of Distribution inside of App Store Connect

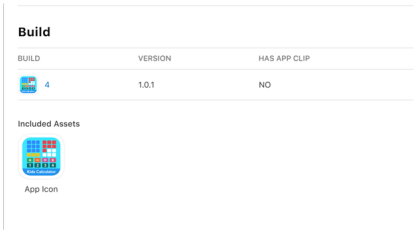

Kids Calculator (94228c)
Distribution
TestFlight
Xcode Cloud
iOS App
1.0 Prepare for Submission
Add Platform
General
App Information
App Review
History
App Store
TRUST & SAFETY
App Privacy
App Accessibility
Ratings and Reviews
GROWTH & MARKETING
In-App Events
Custom Product Pages
Product Page Optimization
Promo Codes
Game Center
iOS App Version 1.0
Save
Add for Review
The assets and metadata below appear on your app's product page, when users install your app, and will be used for web engine search results once you release your app.
English (U.S.)
Previews and Screenshots
Adding accurate screenshots of your app on the newest devices can help you represent the app's user experience. Keep in mind that we'll use these screenshots for all display sizes and localizations. Screenshots are only required for iOS apps, and only the first 3 will be used on the app installation sheets.
iPhone
iPad
Apple Watch
View All Sizes in Media Manager
iPhone
6.5" Display
Drag up to 3 app previews and 10 screenshots here.
(1242 × 2688px, 2688 × 1242px, 1284 × 2778px or 2778 × 1284px)
0 of 3 App Previews | 0 of 10 Screenshots | Choose File | Delete All

My iPhone wasn't the right size for taking the screenshots and uploading them directly so I resized them using https://www.resizepixel.com/edit

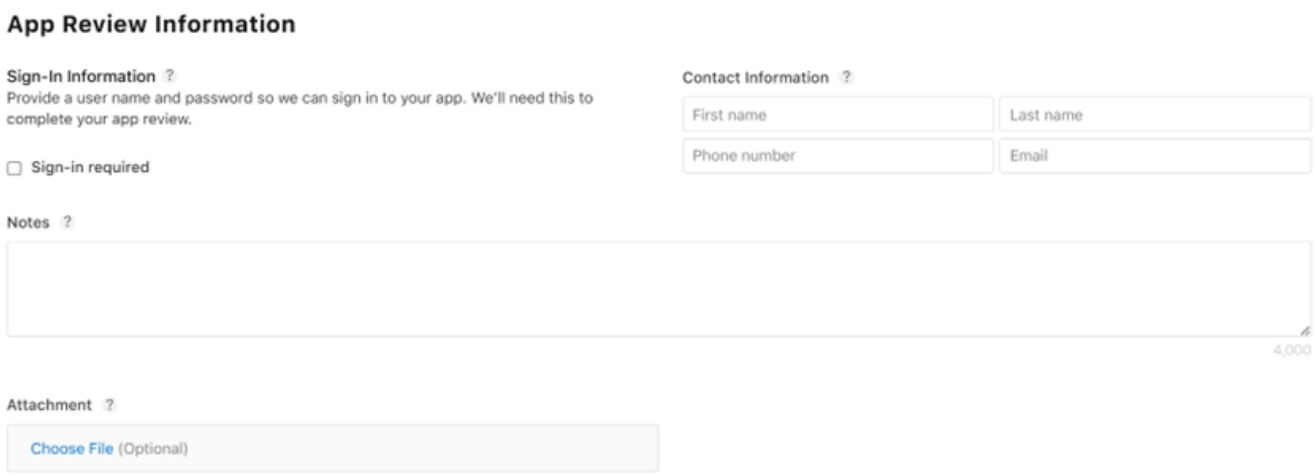
Only one screenshot is required. The App Preview or video is not required


Previews and Screenshots
Adding accurate screenshots of your app on the newest devices can help you represent the app's user experience. Keep in mind that we'll use these screenshots for all display sizes and localizations. Screenshots are only required for iOS apps, and only the first 3 will be used on the app installation sheets.
iPhone
iPad
Apple Watch
View All Sizes in Media Manager
iPhone
6.5" Display
0 of 3 App Previews | 1 of 10 Screenshots | Choose File | Delete All

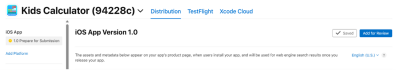
Complete all of the required information and add your build file



If the user does not need to sign in leave this blank.



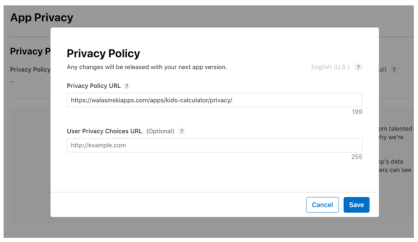
Once finished go to the top and click save then click add for review.



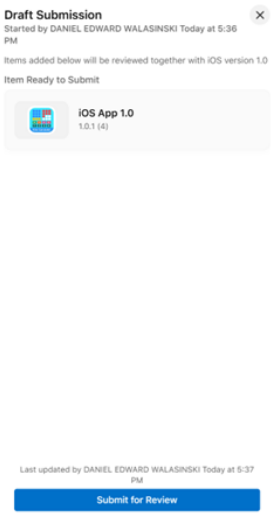
It will give you a list of things you still need to complete.

Make sure you press save after updating each page.

You'll need to add a privacy policy webpage. Here's an example: https://walasinskiapps.com/apps/kids-calculator/privacy/



Once you complete all of the requirements you can submit. The review process takes up to 48 hours.


Draft Submission
Started by DANIEL EDWARD WALASINSKI Today at 5:36 PM
Items added below will be reviewed together with iOS version 1.0
Item Ready to Submit
iOS App 1.0
1.0.1 (4)
Last updated by DANIEL EDWARD WALASINSKI Today at 5:37 PM
Submit for Review

Perhaps after completing this training, you have decided that you want to try coding something exclusively for Apple using Swift and Xcode. After all, Apple products are beautiful, and their user interfaces are too. In many ways, you would be developing for a premium niche market. The table below looks at some of the possibilities.

Swift Development Across Apple Devices			
Device	**OS**	**Common App Types**	**Features**
iPhone	iOS	Messaging, social media, camera apps, maps, fitness, games, shopping, banking, education, travel, AI tools	Camera, microphone, GPS/location, motion sensors, Face ID, Touch ID, push notifications, widgets, Live Activities, Siri/App Intents, Apple Intelligence, ARKit, HealthKit, Apple Pay
iPad	iPadOS	Drawing apps, note-taking apps, classroom tools, music creation, design apps, productivity apps, games	Large touch screen, Apple Pencil, PencilKit, multitasking layouts, keyboard/trackpad support, widgets, Live Activities, handwriting, drawing, document markup
Mac	macOS	Desktop productivity apps, creative tools, video editing, coding tools, games, menu bar utilities, file managers	Keyboard and mouse input, large resizable windows, menus, desktop widgets, file system access, Apple silicon performance, Metal graphics, video processing
Apple Watch	watchOS	Fitness trackers, health apps, timers, habit trackers, sports apps, breathing apps, quick notifications	Heart rate, gyroscope, accelerometer, workout data, route maps, complications, Smart Stack widgets, Bluetooth device connections, haptics, quick interactions on the wrist
Apple TV	tvOS	Streaming apps, media libraries, video players, party games, living-room fitness, video conferencing, family games	Siri Remote, large-screen interface, focus-based navigation, 4K video, HDR, Dolby Vision, Dolby Atmos, Picture in Picture, Continuity Camera with iPhone or iPad

Chapter 14 Publish and Test an App on the Apple App Store Rubric

This chapter focused on preparing an iOS app for Apple App Store submission, creating an iOS build, testing with TestFlight, preparing App Store Connect materials, and documenting the publishing process.

Category	Points	Evidence of Completion
Project Preparation	5	The iOS app project is reviewed and prepared for publishing, including app name, version, bundle identifier, icon, splash screen, and required assets.
iOS Build	5	An iOS build is created or attempted using the required build process. The build status, build file, or build attempt is documented.
TestFlight Testing	5	The app is tested through TestFlight, a simulator, a device, or an approved testing method. Crashes, layout issues, broken buttons, or other problems are identified and revised.
App Store Connect Materials	5	App Store materials are prepared, including app description, screenshots, privacy policy information, app category, age rating, and required metadata.
Evidence of Work	5	Screenshots, build proof, TestFlight proof, App Store Connect draft, privacy policy proof, or other project evidence are submitted.
Reflection	5	A short explanation describes the iOS publishing process, the testing results, and relevant computer science vocabulary such as iOS, build, bundle identifier, TestFlight, App Store Connect, metadata, privacy, testing, or debugging.
Total	**30**	

Submission Checklist

Requirement	Check
App name and version checked	☐
iOS bundle identifier checked	☐
App icon and splash screen checked	☐
iOS build created or attempted	☐
App tested with TestFlight, simulator, device, or approved method	☐
At least one issue or improvement revised	☐
App Store description or listing draft prepared	☐
Screenshots and required metadata prepared	☐
Build proof, testing proof, or App Store Connect proof included	☐
Short reflection completed	☐

Reflection Prompt

Reflection response: 3-5 sentences.

> *Describe the process used to prepare the app for the Apple App Store. Explain one testing result, build issue, or App Store Connect requirement that was checked. Identify one problem, improvement, or publishing detail that was revised before final submission.*

15
Resources and References

The links below are official tools, documentation pages, and learning resources used throughout this book.

Code Editors and Development Environments

- Cursor — https://cursor.com
- Visual Studio Code — https://code.visualstudio.com
- GitHub Codespaces — https://github.com/features/codespaces
- Xcode — https://developer.apple.com/xcode/

Web and Mobile App Development

- CodeSandbox — https://codesandbox.io
- Node.js — https://nodejs.org
- Expo — https://expo.dev
- Expo Documentation — https://docs.expo.dev
- Expo Snack — https://snack.expo.dev
- React Native — https://reactnative.dev
- Flutter — https://flutter.dev
- FlutterFlow — https://flutterflow.io

Design Tools

- Figma — https://figma.com
- Canva — https://canva.com
- RealFaviconGenerator — https://realfavicongenerator.net

Publishing Platforms

- Google Play Console — https://play.google.com/console

- Apple Developer — https://developer.apple.com
- App Store Connect — https://appstoreconnect.apple.com
- TestFlight — https://developer.apple.com/testflight/

Additional Development Tools

- Python — https://www.python.org
- PyInstaller — https://pyinstaller.org
- Android Developers — https://developer.android.com
- Microsoft Learn — https://learn.microsoft.com
- Electron — https://www.electronjs.org
- Tauri — https://tauri.app